NTIS

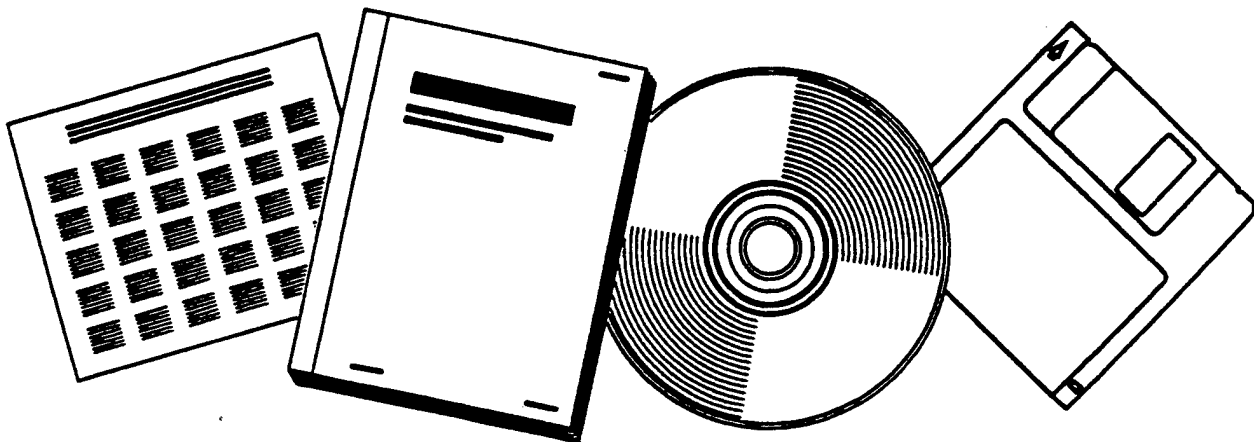# EXPLORING THE INTERACTION OF GEOMETRY AND SEARCH IN PATH PLANNING

DTIC QUALITY INSPECTED 3

STANFORD UNIV., PALO ALTO, CA

19970702 030

FEB 92

U.S. DEPARTMENT OF COMMERCE
National Technical Information Service

Exploring the Interaction of Geometry and Search in Path Planning

by

David J. Zhu

Department of Computer Science

Stanford University
Stanford, California 94305

REPORT DOCUMENTATION PAGE

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE February 1992	3. REPORT TYPE AND DATES COVERED

4. TITLE AND SUBTITLE

EXPLORING THE INTERACTION OF
GEOMETRY AND SEARCH IN PATH PLANNING

5. FUNDING NUMBERS

6. AUTHOR(S)

David J. Zhu

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Computer Science Department
Stanford University

8. PERFORMING ORGANIZATION REPORT NUMBER

STAN-CS-92-1413

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

DARPA

10. SPONSORING / MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION / AVAILABILITY STATEMENT

12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)

This thesis addressed the problem of developing path planning algorithms that are both efficient and well-behaved. We proposed a novel approach in which we solve a path planning problem by finding and solving an appropriate abstraction of the original problem. We argued that in order for this approach to be efficient, a tighter integration of geometric reasoning and search is essential. This thesis developed evidence, both theoretical and experimental, to support this argument. In particular, we investigated in-depth two approaches for generating problem abstractions: the constraint approximation approach (in the context of robot motion planning); and the problem decomposition approach (in the context of pipe routing). For each of these two approaches, we developed algorithms that tightly integrate geometric reasoning with search and we addressed many issues raised by this integration. These algorithms have been implemented and tested in a robot motion planning system and a pipe routing system.

14. SUBJECT TERMS

15. NUMBER OF PAGES

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
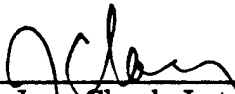Prescribed by ANSI Std. Z39-18

EXPLORING THE INTERACTION OF
GEOMETRY AND SEARCH
IN PATH PLANNING

A DISSERTATION

SUBMITTED TO THE DEPARTMENT OF COMPUTER SCIENCE

AND THE COMMITTEE ON GRADUATE STUDIES

OF STANFORD UNIVERSITY

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

DOCTOR OF PHILOSOPHY

By

David J. Zhu

February 1992

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a dissertation for the degree of Doctor of Philosophy.
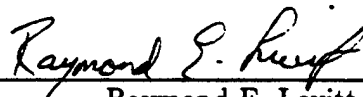
_____
Jean-Claude Latombe
(Principal Adviser)

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a dissertation for the degree of Doctor of Philosophy.

_____
Michael R. Genesereth

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a dissertation for the degree of Doctor of Philosophy.

_____
Raymond E. Levitt
(Civil Engineering)

Approved for the University Committee on Graduate Studies:

_____
Dean of Graduate Studies

iii

Abstract

Planning a continuous path for a rigid object or a collection of rigid objects (which are often referred to as "robots") among obstacles has been widely studied over the last 15 years. In recent years, path planning problems have found a variety of applications ranging from robot navigation to concurrent design. The need to integrate path planning capabilities into a large complex system has placed strong emphasis on the efficiency and reliability of path planning methods. This thesis addresses this need by focusing on the issues that underlie the development of a path planning system that is both efficient and reliable.

Since there is strong indication that path planning is intrinsically complex, rather than trying to develop a faster path planning algorithm, our approach is to find and solve an appropriate abstraction of the original problem that captures the important information while ignoring the irrelevant details. This is achieved by searching a space of possible problem abstractions.

A robot path planning problem usually consists of finding a continuous, finite-length curve connecting two points in a continuous space, called the robot's configuration space. The first step in solving this problem is to transform it into a discrete state-space search problem, where each state is a subset of "equivalent" robot's configurations. In the second step, the state space is searched for a sequence of adjacent states connecting an initial state containing the robot's initial configuration to a goal state containing the robot's goal configuration. The first step deals mainly with geometry while the second step deals with search. In the traditional approaches to path planning, these two steps are considered separately.

In order to search the problem abstraction space efficiently, we must take advantage of the information that can be obtained from an unsuccessful attempt to solve a previous problem abstraction to guide the generation of the next problem abstraction and to help solve it. The implication, we argue, is that the interaction between geometry and search is of critical importance. This thesis is devoted to the exploration of the interaction of geometry and search in the context of path planning. We investigate two complementary approaches for generating problem abstractions: *constraint approximation* and *problem decomposition.*

In the constraint approximation approach, an abstraction of the problem is generated by approximating the constraints defining the robot's free space. This approach addresses the concern that path planning problems require polynomial time in the number and the degree of the constraints applying to the robot's motions. The constraint approximation approach allows us to ignore irrelevant details of these constraints.

In the problem decomposition approach, the original problem is decomposed into a set of subproblems, each involving a lower-dimensional configuration space. This approach derives from the strong evidence that path planning problems are exponential in the dimension of the robot's configuration space. The problem decomposition approach allows us to ignore unimportant interaction among the subproblems.

Even though many of these ideas are not new in Artificial Intelligence (AI) research, this thesis is the first systematic attempt to explore them in depth in a geometric context which is significantly different from the traditional AI domain. The application of these approaches to path planning problems raises many issues concerning the interaction of geometrical algorithms and search algorithms that have not been addressed previously. The investigation of these issues forms the core of this thesis.

In this thesis, the constraint approximation approach is studied in the context of robot navigation problems. The techniques developed for this approach are implemented in a robot path planner. The problem decomposition approach is investigated

in the context of pipe routing problems. A pipe router has been implemented that incorporates the techniques developed for this approach.

Acknowledgements

I am indebted to my thesis advisor, Jean-Claude Latombe, for his guidance and support that has made this research possible. His commitment to his students and research has been a continual source of inspiration for me.

I would like to thank the other members of my reading committee and my defense committee, Mike Genesereth, Ray Levitt, Yoav Shoham, and Mark Cutkosky, for their helpful commands.

I am grateful to the members of the "reasoners" group of the Robotics Laboratory for providing a nurturing research environment. In particular, I would like to thank Wonyun Choi and Shashank Shekhar for many fun and helpful discussions.

I would like to thank my business partner, Jim Slater, with whom I have founded Nomadic Technologies, and other members of the company for their understanding when I took time off to finish my thesis.

Being far away from home, I am very fortunate to have the loving families of my relatives in America. I especially like to thank Aunty Jessie, Uncle Ben, Aunty Jean, and Uncle Percy for their love and support.

Finally, I would like to thank my parents for believing in me. They have sacrificed selflessly to give me the opportunity and the freedom to pursue my dream. Without their love and guidance, this would not have been possible.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivation and Goal

Planning a continuous path for a rigid object or a collection of rigid objects (which are often called "robots") among static obstacles has been widely studied over the last 15 years. It is a problem with many potential applications in robotics, design, process planning, simulation, and graphic animation. This problem is often referred to as the "piano-mover" or the "sofa" problem.

Two very different approaches have been developed to solve this problem [Latombe, 1991]. One consists of first capturing the connectivity of the collision-free subset (the "free space") of the robot's configuration space[1] into a "connectivity graph" and next searching this graph. Along this general approach, "cell decomposition methods" proceed by decomposing the free space into simple regions called cells and building the connectivity graph that represents the adjacency relation among these cells [Schwartz and Sharir, 1983a and 1983b] [Brooks and Lozano-Pérez, 1985]. Along the same approach, "roadmap methods" reduce the free space to a network of one-dimensional curves (the "roadmap") yielding the connectivity graph [Ó'Dúnlaing,

---

[1] The concept of configuration space is well known in the robotics literature. In order for the thesis to be self-contained, a brief definition and description of this concept will be given in Subsection 2.1.2 (Chapter 2).

Sharir and Yap, 1983] [Canny and Donald, 1988]. The second approach to path planning consists of defining a function over the robot's configuration space with a global minimum at the goal configuration and tracking the steepest descent of this function. The function is often interpreted as a potential field, and the robot configuration as a particle under the influence of this field, so that the methods using this approach are called "potential field methods" [Khatib, 1986].

The first of these two approaches tends to spend a large portion of computing time constructing the connectivity graph. In contrast, the second approach often needs almost no precomputation before its actually starts searching for a path. This is due to the fact that the potential function is both defined and computed locally while its steepest descent is being tracked. The problem, however, is that local definition and computation yield the possibility of local minima in which path planning may get trapped. For this reason, some potential field methods use the potential function as a heuristics to guide the search of a grid placed over the configuration space and allow some form of backtracking to escape from local minima [Barraquand and Latombe, 1991]. These methods may be efficient in practice, but their worst-case behavior is known to be poor and their actual behavior is extremely difficult to characterize in advance. In particular, while they may give good results in apparently difficult situations, they may also be quite disappointing in simple ones. In addition, they are in general unable to properly detect that no path exists. Indeed, before they can make such a decision, they in general have to fully explore the configuration space.

A few researchers have attempted to develop methods for computing potential functions with no minima other than the goal configuration [Koditschek, 1987] [Rimon and Koditschek, 1990], but such functions, which are called "navigation functions", can no longer be defined locally and their computation is somewhat equivalent to that of capturing the connectivity of the free space in the form of a graph. Hence, it requires substantial precomputation.

We believe that path planning methods with well-defined properties will be needed more and more in order to incorporate them in large application systems (e.g. autonomous robots, CAD systems, process planners). On the other hand, these methods

will also have to be fast. Indeed, in an autonomous robot, they will have to be applied in quasi-real-time, while in a CAD system or in a process planner, they will have to be applied repetitively without substantial degradation of the response time to the user. These two goals – being well-behaved and being fast – are somewhat antagonistic. Even though they have been achieved separately, there does not exist as yet a path planning method which achieves both goals at the same time.

We feel that we are more likely to achieve these two goals simultaneously by using planning methods that build and exploit an explicit, yet condensed representation of the free space in the form of a connectivity graph, than by using methods that don't. Indeed, the latter is intrinsically shortsighted. Although improvements of potential field methods are certainly possible (for example, in [Barraquand and Latombe, 1991], a fast, well-contained precomputation in the workspace abstracts useful global topological information about the free space), it seems that it is an inherent part of these local methods to have poorly characterized behaviors.

Our research addresses the efficiency issue in path planning with explicit representation of the free space.

1.2 Approach

1.2.1 Searching in a Space of Abstraction Problems

One approach to addressing the efficiency issue in path planning is to attempt to produce a faster path planning algorithm. However, there are strong indications that path planning is intrinsically exponential in the dimension of the robot's configuration space (number of degrees of freedom) and polynomial in both the number of constraints defining the boundary of the free space and their maximal degree (assuming without loss of generality that these constraints are algebraic). This has not been formally proven, but path planning with arbitrarily many degrees of freedom has been shown to be PSPACE-hard [Reif, 1979] [Hopcroft, Schwartz and Sharir, 1984], and all existing algorithms that can solve path planning problems for robots with arbitrarily many degrees of freedom have exponential worst-case complexity in

the robot's number of degrees of freedom. All methods that use algebraic models to represent the robot and the obstacles have polynomial complexity in the number and the degree of the algebraic equations that define the boundary of the free space.

Another approach to efficiency is to attempt to find and solve an appropriate approximation of the original problem. This is accomplished by considering successive abstractions of the original problem, with each abstraction being more accurate than the previous one. Since the complexity of the constraints and the dimension of the configuration space are the two main parameters influencing the running time of a planner, we may reasonably expect to reduce the average computing time by approximating the constraints into simpler ones and/or breaking the problem into several subproblems, each with a lower-dimensional configuration space. Indeed, if a robot has to navigate between two offices in one wing of a building, it probably does not have to consider the exact shape of the objects located in another wing, although in exceptional cases these objects might become obstacles to consider (for instance, if the building is so cluttered that the robot has to navigate through the other wing). In a similar fashion, if two three-degree-of-freedom mobile robots navigate in the same building, they form a six-degree-of-freedom robotic system. But, rather than planning the path of such a system in a six-dimensional configuration space, it seems reasonable to break this problem into two planning problems, each occurring in a three-dimensional space. This corresponds to considering each robot separately, with the other being a potential moving obstacle. Except in rare cases where, for example, the two robots would navigate in opposite directions in a narrow corridor, the two problems can be solved quite independently. These two examples illustrate that not all the constraints in a problem are relevant to finding a solution of that problem. Ignoring irrelevant information by abstracting the original problem into a simpler problem may allow a planning algorithm to find a solution more efficiently.

Therefore, rather than looking for a new and better path planning algorithm, the approach explored in this thesis is to find an appropriate abstraction of the original problem that can be solved easily. This abstraction should admit at least one solution

if solutions exist for the original problem. Furthermore, the solutions of the abstraction problem should form a subset of the solutions of the original problem. Finding such an abstraction is accomplished by generating and solving successive abstractions of the original problem until a solution has been found or it has been shown that no solution exists. This process can be viewed as searching in a space of abstraction problems. Of course, it is efficient only if the cost of searching in the problem space (which is equal to the sum of the costs of generating and solving several abstraction problems) is less than the cost of solving the original problem.

This general approach is certainly not new. The use of abstraction in hierarchical planning has been suggested many times in Artificial Intelligence (e.g. [Sacerdoti, 1974], [Tenenberg, 1988], [Unruh and Rosenbloom, 1989], [Korf, 1980], [Knoblock, 1990], [Christensen, 1990]). However, in motion planning, although the hierarchical approach has often been suggested, it has rarely been put to practice (e.g. [Faverjon, 1984]). In fact, the actual implications of reasoning in a hierarchy of abstraction problems on path planning algorithms have never been investigated in depth before. As we discuss in the next subsection, classical path planning algorithms separate the considerations of geometry and search. Pushing the hierarchical approach beyond its superficial intuitive idea leads to developing new algorithms where geometry and search strongly interact. Most of our research has been devoted to the investigation of this interaction.

1.2.2 Interaction of Geometry and Search

As mentioned above, a typical cell decomposition or roadmap path planning algorithm operates in two steps: the first deals with geometrical constraints and captures the connectivity of the free space into a connectivity graph; the second searches this graph. These two steps are considered separately, and the only interaction between them is when the first transfers the constructed connectivity graph to the second. The input problem is geometrical in nature, hence continuous, and the role of the first stage is to *discretize* this problem, so that it becomes solvable by combinatorial search techniques.

Notice the difference with classical planning in Artificial Intelligence. In AI, the input problem is discrete; the state space that will have to be searched is defined by an input initial state and input operators that map states into states. However, in motion planning, states are not given in advance. It is the task of the planner to extract them from the input geometry of the robot and the obstacles. For example, in a cell decomposition method, extracted cells are subsets of the free space such that any two configurations in the same cell can easily be connected by a path. Hence, with respect to path planning, a cell may be regarded as a subset of equivalent configurations, that is, as a state. When geometry and search are separated, as in the case of a classical path planner, the state space is fixed throughout search.

The main implication of building a path planner that attempts to solve a problem by solving successive abstractions of this problem is that geometry and search can no longer be considered separately. Indeed, suppose that the core path planning algorithm used in this planner fails to find a path for some abstraction problem, say, because the constraints defining the free space have been too grossly approximated. In such a situation, it would be quite inefficient to blindly generate a new abstraction problem and solve this problem from scratch. We would like, instead, for the failed search to return not just failure, but also to return pertinent information about how to refine the problem in order to reduce the chances of failing again. This information could then be used to guide the search of the problem space.

In addition, when the core planning algorithm fails to solve an abstraction problem, we would like this problem to be refined in such a way that the discretization of the new problem is directly related to the discretization of the problem whose solution failed, so that the new connectivity graph is an expansion of the previous one and the new search can take advantage of the search work done previously. Talking in term of states, this means that the new discretization should essentially be obtained by refining some states, for instance by decomposing them into sub-states, keeping most transitions between states unchanged, rather than by building a completely new set of states that would not be related to the states extracted from the previous problem.

Therefore, efficiency within the problem approximation approach requires that the

algorithms manipulating the input geometrical constraints and discretizing the path planning problems interact strongly with those performing graph searching. As we will see in Section 1.3, this interaction raises many important issues, most of which have not even been suggested in the brief discussion above. Our research has been devoted to the exploration of these issues. Although the idea of searching a space of abstraction problems has been the starting point of our research in path planning, the exploration of the interaction between geometry and search, which is induced by this idea, has been the main topic of our research, and it has become the key element of our approach to path planning. We feel that the interaction between geometry and search will become an increasingly important aspect in the development of large computer systems performing geometrical reasoning (path planning being one particular case of geometrical reasoning). We believe that the issues explored and the results obtained in this thesis are relevant to other geometric problem areas targeted by these systems.

1.2.3   Approximate Cell Decomposition

Our initial idea was to use a "core" path planning algorithm and to embed it in a larger system. While the core algorithm would have manipulated geometric constraints and searched connectivity graphs for paths, the larger system would have searched a space of abstraction problems. However, as we have discussed above, the hierarchical planning approach affects the core algorithm itself by requiring that geometrical computations and graph searching operations be integrated more tightly than they are in classical path planning algorithms. One might imagine that this interaction is achievable by decomposing the core algorithm into modules and by slightly changing the overall control structure to accommodate the needs of the overall problem approximation approach. If this had been possible, we could have conducted our research quite independently of the planning method used by the core algorithm. However, as explained below, the problem approximation approach affects the very nature of the core algorithm.

As an illustration, let us consider an "exact" cell decomposition method such as the ones proposed in [Schwartz and Sharir, 1983a] and [Schwartz and Sharir, 1983b].

We assume that the robot and the obstacles are defined as semi-algebraic sets. Given a planning problem, we may generate an abstraction problem by simplifying the shapes of the robot and the obstacles. This simplification yields simpler equations defining the robot's free space. Let m be the dimension of the configuration space. An exact cell decomposition method typically builds a cylindrical decomposition of the free space into cells by computing noncritical regions in a $(m-1)$-dimensional projection of the configuration space (in order to do so, it recursively computes noncritical regions in a $(m-2)$-dimensional projection, and so on), and then lifts the regions into cells of the m-dimensional configuration space. Ultimately, one gets a representation of the free space as a collection of cells, each defined as a semi-algebraic set homeomorphic to a ball in $\mathbf{R}^m$. Path planning then consists of constructing and searching the graph of the adjacency relation among these cells (i.e. the connectivity graph inferred from the decomposition). The cylindrical nature of the decomposition allows the planner to transform any path extracted from this graph into a continuous path for the robot.

Now, in the context of the hierarchical planning approach, if the search of the above connectivity graph fails, one faces (at least) three difficulties:

- First, no simple information is available to help determining how the shapes of the robot and the obstacles have to be refined in order to generate a new abstraction problem.

- Second, if a new abstraction problem is generated, the decomposition process does not guarantee that the new decomposition is a refinement of the previous decomposition.

- Third, as long as the planner has not tried to solve the original problem, it has no way to decide that this problem admits no solution, when this is the case.

The first of these difficulties results from the fact that the "unknown" regions (the regions for which we don't know whether they are in the free space or not) of the configuration space are not represented explicitly because we have approximated the shapes of the robot and the obstacles separately. The second difficulty is caused by the rather complicated shapes of the cells and the involved process that constructs

them. The third difficulty is related to the first, and is also due to the fact that the planner has no explicit representation of a region that is guaranteed to be a subset of the obstacle region.

Although there might be ways to alleviate, or perhaps even eliminate, these difficulties, another type of planning method, known as the "approximate" cell decomposition method seems more appropriate to handle them [Brooks and Lozano-Pérez, 1985] [Gouzénes, 1984] [Laugier and Germain, 1985] [Faverjon, 1986] [Kambhampati and Davis, 1986]. Basically, this method consists of decomposing the robot's configuration space into rectangloid cells[2]. Cells are classified as EMPTY or FULL, depending on whether they lie entirely outside or entirely inside the obstacle region in the configuration space. If they are neither EMPTY, nor FULL, they are classified as MIXED. The cells are generated and labeled using the input shapes of the robot and the obstacles. The resulting cells define an approximation of the input problem. The union of the EMPTY cells forms a conservative approximation (i.e. a subset) of the free space. The MIXED cells form "unknown" regions. The union of the EMPTY and MIXED cells forms a superset of the free space. The planner first searches the connectivity graph of these cells for a sequence of adjacent EMPTY cells connecting the initial configuration to the goal configuration. If a sequence is found, a path is easily generated from it. If no sequence is found, it instead tries to generate a sequence of EMPTY and/or MIXED cells. If such a sequence exists, the MIXED cells contained in it provide a nice hint toward where to look in order to generate the next abstraction problem. Indeed, one simple way to proceed is to decompose the MIXED cells in the sequence into smaller rectangloid cells and label the newly generated cells. Decomposing a rectangloid cell into smaller rectangloid cells corresponds to exploding a node of the connectivity graph into multiple nodes, without affecting the rest of the graph. If no EMPTY/MIXED cell sequence exists connecting the initial configuration to the goal configuration, it is guaranteed that no free path exists; hence, the planner can safely return failure without having to solve the original problem.

---

[2]The cells need not be rectangloid. But they should have certain properties that guarantee that the generation of a path within a cell is straightforward. Rectangloid cells possess such properties. Another such type of cells, called "slippery cells", is proposed in [Quinlan, 1991].

Therefore, the approximate cell decomposition approach handles the three difficulties listed above as follows:

- By explicitly representing "unknown" regions of the configuration space as MIXED cells and, when needed, including these cells in the connectivity graph, it helps in discovering which constraints should be refined.

- By having all cells share the same basic shape (a rectangloid), it makes it easy to refine an abstraction problem in such a way that the new search graph is an expansion of the previous one.

- By having an explicit representation of the forbidden region in configuration space (FULL cells), the method may detect that no solution exists for the original problem at an early stage of the abstraction process.

Because it seemed so much more compatible with a hierarchical path planning approach than any other existing method, we have chosen to use the approximate cell decomposition method as the core algorithm of our planning system.

1.2.4 Summary

In summary, the approach to path planning developed in this thesis is based on the following arguments and ideas:

- It is important to produce path planners that are both efficient and well-behaved from the computational point of view.

- An explicit, global representation of the connectivity of the free space is a key element toward achieving well-behavedness in path planning systems.

- Due to the intrinsic complexity of path planning, we propose a problem approximation approach that consists of searching a space of abstraction problems.

- Efficiency of searching the space of abstraction problems depends on a tighter integration between geometry and search.

- Approximate cell decomposition provides a promising basis to achieve this interaction.

1.3 Overview of Issues and Results

We propose two different, but complementary, ways to generate abstraction problems in path planning:

- One is *constraint approximation*. It consists of generating an abstraction of a problem by approximating the constraints defining the robot's free space.

- The other is *problem decomposition*. It consists of decomposing a problem into subproblems, with each subproblem involving a lower-dimensional configuration space than the original problem.

Constraint approximation is related to the fact that path planning is polynomial in the number and the degree of the algebraic equations defining the free space boundary. Problem decomposition is related to the strong presumption that path planning requires exponential time in the dimension of the configuration space.

These are two very general ways of simplifying a problem. Nevertheless, their application to path planning raises many issues related to the interaction between geometry and search, whose in-depth analysis has not been made before. This section provides an overview of the issues and the results that will be presented in more detail in the other chapters of this thesis.

1.3.1 Constraint Approximation

In the constraint approximation approach, we have investigated two classes of issues: representation of constraints and hierarchical search. The first issue deals with the efficient generation of an abstraction of the problem, while taking advantage of the information provided by the constraints. The second issue concerns the efficient generation of a solution to the abstract problem (or the detection of the absence of a solution), while taking advantage of the previous search effort, if any. We discuss

them in more detail below.

Constraint Reformulation   The boundary of the robot's free space is made of configurations where the robot is in contact with one or more obstacles in the workspace, with no overlapping of the objects' interiors. Its representation in the form of equations can be derived from the equations representing the boundaries of the robot and the obstacles [Latombe, 1991]. In general, the equations representing the free space determine complicated pieces of a curved surface, even when the shapes of both the robot and the obstacles are fairly simple. These equations are the constraints within which the robot's motions should occur. The purpose of approximating these constraints is to simplify the problem of finding paths in the free space.

The approximate cell decomposition method represents the configuration space as a collection of rectangloid cells labelled as EMPTY, MIXED, or FULL. The union of the EMPTY cells is a subset of the free space, while the union of the EMPTY and MIXED cells is a superset of the free space. The boundaries of these two unions are approximations of the constraints on the robot's motions in the form of collections of rectangular faces.

When we construct a rectangloid decomposition of the configuration space, we should try to satisfy the following two goals, simultaneously:

- The number of cells in the decomposition should be minimized, in order to keep the size of the connectivity graph as small as possible. This goal directly relates to the motivation of the hierarchical strategy: we want the planner to consider "details" only when necessary.

- The volume of the region formed by the EMPTY and FULL cells should be maximized. This goal relates to reducing, as quickly as possible, the "uncertain" region, i.e. the MIXED cells. Clearly, decomposing a MIXED cell into smaller cells would not be useful if all of these new cells were MIXED.

These two goals are somewhat conflicting since one obvious way to achieve the second one is to produce many small cells, which contradicts with the first one. In

addition to these two goals, the computational cost of generating the decomposition should remain as small as possible.

A simple method of decomposing the configuration space into labeled rectangloid cells is to recursively partition a cell (initially the whole configuration space is considered as a cell) into 2^m cells of equal size where m is the dimension of the configuration space [Faverjon, 1984] [Ayala et al., 1985] [Kambhampati and Davis, 1986]. This method is called the "quadtree" method or the "octree" method when $m = 2$ or $m = 3$, respectively. Even though this method leads to a decomposition easily representable in a tree structure of degree 2^m, it tends to produce a large number of MIXED cells. This method belongs to a class of cell decomposition methods that we call "divide-and-label" methods (Subsection 2.2.2).

We have developed a new decomposition method based on constraint reformulation, which provides significantly better experimental results than earlier decomposition techniques. We call our method an "approximate-and-decompose" method because, in the first phase of processing, it approximates every individual obstacle in the configuration space as a collection of rectangloids, and, in the second phase, it computes the complement of all these rectangloids, yielding the full decomposition of the configuration space. Two types of approximation are used: *bounding* and *bounded* approximations. Unlike the divide-and-label methods, the labelling of cells is a direct by-product of the decomposition algorithm and requires no additional computation. Experimental results show that, compared to the other approximate cell decomposition methods, our method generates fewer cells and the number of cells it generates grows slower when the shape of the free space becomes more complicated.

Hierarchical Search After a decomposition of the configuration space has been computed, a connectivity graph can be generated and searched using the EMPTY cells. Since the union of these cells forms a conservative approximation of the free space, if a path is found, it is actually a free path, i.e. a solution path. However, the connectivity graph may not contain a solution path, in which case the decomposition should be refined.

Refining the current decomposition of the configuration space can be achieved by decomposing some MIXED cells into smaller rectangloid cells and labelling them appropriately. The first question that arises is: which MIXED cells should be refined? Refining all of them might be quite costly and not very relevant. A better approach is to generate and search a connectivity graph using both the EMPTY and MIXED cells; if a path (sequence of cells) is found in this graph, it focuses its attention on the subset of MIXED cells that belong to this path for refinement. Now, assuming that these cells are refined, a new question arises: how do we take advantage of the unsuccessful search to guide the search of the new connectivity graph? Indeed, since the new graph is obtained by breaking down some nodes representing MIXED cells into nodes representing smaller cells, most of the connectivity graph remains unchanged and it would be a waste of computing time to search it again from scratch. One way to proceed is to consider the problem of finding a path through each decomposed MIXED cell as a subproblem of the original problem and to solve it by searching the connectivity subgraph generated using the cells resulting from the decomposition of this MIXED cell. But then, how do we combine the solutions of the subproblems together to obtain a solution of the whole problem? And if we fail to solve some of these subproblems (perhaps because we focused our attention on the wrong MIXED cells), how do we decide that we may be exploring a dead-end and that we should shift our attention toward other MIXED cells? If this happens, we may have to search a previously searched connectivity graph for a new path, raising yet another question: while searching a connectivity graph for a path, is there information that we can remember which can be used to accelerate a future search of the same graph? For example, if we discover that two EMPTY cells cannot be connected through a MIXED cell that is adjacent to both of them, it is probably a good idea to remember that fact and not to explore that same connection again.

We have developed a hierarchical graph searching approach that embodies responses to the above questions. This approach consists of generating a hierarchy of connectivity graphs. Each graph represents a one-level decomposition of either the configuration space or a MIXED cell. Compatibility constraints are associated with each connectivity graph to guarantee that the solutions generated in the various

graphs match each other to provide a solution of the original problem. Furthermore, while they are searched, nodes of these graphs are annotated with expressions summarizing interesting connectivity properties which can be used to facilitate the search of the same graphs in the future. These annotations are reminiscent of the NOGOOD expressions used in truth maintenance systems [Doyle, 1979].

1.3.2 Problem Decomposition

Problem decomposition consists of generating an abstraction of a path planning problem by decomposing it into subproblems, with each subproblem having a lower-dimensional configuration space than the original problem. The key concern here is the interaction of the subproblems. If the interactions among the subproblems are weak enough, we may expect that, most of the time, we can solve these problems independently and combine their solutions to get a solution of the whole problem. However, the subproblems are usually not completely independent, and it may happen that the generated solutions may not fit together to provide an acceptable solution. The main issue is how to backtrack intelligently to handle harmful interaction among subproblems. If subproblems interact very strongly, solving them in a sequential fashion may cause excessive backtracking. In this case, another question is: Can we consider the interaction among subproblems early enough, i.e. before we commit to a specific solution for each of them, so that we avoid or reduce the need for backtracking? Both these two cases are discussed in greater detail below.

Backtracking Let us assume that we solve subproblems sequentially, one at a time. We use the solution generated for every subproblem as a source of additional constraints for subsequent subproblems. For example, when planning the paths of n robots, we may consider them in sequence. Once a path is found for a robot, it is considered as a moving obstacle by subsequent robots. Treating solutions to the solved subproblems as constraints for the not yet solved subproblems is one way to handle interactions among subproblems. It works well as long as we succeed in solving each subproblem sequentially. It may happen, however, that a subproblem becomes over-constrained. This is discovered when we fail to find a solution to one of the

subproblems. As we will see in more detail in this thesis, some interactions among subproblems are reminiscent of the interactions that may occur among subgoals in classical AI planning (e.g. [Sussman, 1975] [Nilsson, 1980]).

One way to recover from the failure to solve a subproblem is to backtrack to a previously solved subproblem and to solve it in a different fashion. Indeed, a different solution may yield different constraints for subsequent subproblems. However, blind chronological backtracking would be quite inefficient since it could often lead to changes that are irrelevant to the failure. Therefore, the following questions arise naturally. How can we identify the subproblems whose solutions are responsible for over-constraining the subproblem that we just failed to solve? Next, assuming that we know how to identify these subproblems, how should we modify their solutions in a way that is relevant to the encountered failure? Finally, what should we remember so that we do not loop through a cycle of failures? Similar questions arise in logic-based problem solvers (e.g. [Stallman and Sussman, 1977] [Latombe, 1979]), but now we have to answer them in a geometric context. This context makes them especially difficult because every path planning problem may admit a continuous infinity of solutions. We cannot examine all of them, and a tradeoff between efficiency and completeness of the backtracking algorithm is probably necessary.

We have developed new selective backtracking techniques for dealing with interactions among subproblems. Interestingly, these techniques are mainly built upon the same basic geometric algorithms that we use to solve a subproblem. For instance, when we fail to solve a subproblem, we attempt to solve it without considering the constraints imposed by the previously solved subproblems. This simply leads to adding cells in the connectivity graph of the current subproblem. If a solution (a sequence of cells) is found, those cells contained in this solution which have just been added to the connectivity graph allow us to identify previous solutions that are relevant to the failure. On the other hand, if no solution is found with this extended graph, it just means that the encountered failure is independent of the way in which previous subproblems have been solved, and the planner can return failure for the whole problem. In a way comparable to some AI planners (e.g. [Warren, 1974] [Waldinger, 1977]), our

planner may protect generated solutions from being modified by subsequent failures in order to avoid infinite looping.

Lesser Commitment   Experiments have shown that our backtracking techniques work efficiently as long as the subproblems interact weakly, which is often the case in multi-robot path planning. However, when this interaction increases, excessive back-tracking becomes the main source of inefficiency. In multi-robot planning problems, this occurs when we increase the number of robots in the same workspace. When the interactions among subproblems become too strong, the decomposition into subproblems is probably no longer justified, and we should rather try to solve the original problem as a whole (that is, by considering a high-dimensional configuration space). Before this extreme situation is reached, however, intermediate situations may be handled by other techniques.

One of the problem with solving subproblems in sequence is that it may lead to committing ourselves too early to specific solutions for some subproblems. In multi-robot planning, for example, planning the paths of the first few robots may be very under-constrained, and choosing a particular path for each of them may unduly over-constrain the paths of the other robots. One way to avoid excessive backtracking is to avoid this early commitment, until sufficient information is made available to select specific paths reliably. This corresponds to applying a least-commitment strategy to the problem of dealing with interactions among subproblems.

However, although the least-commitment idea is intuitively attractive, its application to path planning problems is far from straightforward. It raises two basic questions. When we consider subproblems sequentially and we generate a specific solution for each of them, it is not too difficult to represent the constraints that this solution applies to the subproblems that are still pending. But, if we avoid committing ourselves to specific solutions, how should these constraints be generated and represented? And next, how should the consistency among the constraints be checked and enforced? Indeed, it is well-known that, while "2-consistency" can be checked rather efficiently, verifying full consistency requires exponential time in the number of

subproblems (i.e. the number of paths in our case) [Mackworth and Freuder, 1984]. This is equivalent to planning paths for multiple objects in the Cartesian product of the configuration spaces of these objects.

To address these questions, we have developed a lesser-commitment approach that combines local consistency checking algorithms and the selective backtracking method discussed earlier. In this approach, we also consider the subproblems sequentially. For each subproblem, the approximate cell decomposition method gives us a set of solutions in the form of a sequence of EMPTY cells which we call a "channel". The difference is that we do not transform this channel into a specific path immediately. This means that we only commit ourselves to a class of solutions represented by the channel, but not to a single path in this channel (hence, the term "lesser-commitment"). After finding channels for all subproblems, we analyze the interactions among these channels. Intuitively, two channels interact if they contain paths along which the corresponding robots collide. When an interaction is detected, we locally reduce the size of the channels. If the interaction cannot be removed, backtracking is invoked to generate an alternative channel.

1.3.3 Summary

In this thesis we investigate two different, but complementary, approaches for generating abstractions of path planning problems:

- *constraint approximation* generates an abstraction of a path planning problem by approximating the constraints defining the robot's free space.

- *problem decomposition* generates an abstraction of a path planning problem by decomposing it into a number of subproblems each with a lower-dimensional configuration space.

The key technical issues explored in this thesis are:

- approximate cell decomposition,

- hierarchical search,

- selective backtracking,

- lesser-commitment.

1.4 Applications

Path planning has many potential applications in robotics, design, simulation and graphic animation. In this thesis we illustrate the methods that we have developed in the context of two applications: mobile robot navigation (motion planning) and pipe layout design (pipe routing). With respect to mobile robot navigation we have developed a path planner for a rigid object that translates and rotates in the plane. This planner generates successive approximations of the constraints on the robot's motion in its configuration space. It incorporates the techniques introduced in Subsection 1.3.1. With respect to pipe layout design, we have developed a sequential pipe router and a parallel pipe router. Both routers cope with the high-dimensionality of the problem by dividing it into a set of subproblems each consisting of the routing of one pipe. These routers incorporate the techniques introduced in Subsection 1.3.2.

1.4.1 Mobile Robot Navigation

An autonomous robot must be equipped with a navigation system that plans and monitors the motions of the robot. Such a system typically focuses on how to make the robot achieve a specified goal configuration from its current initial configuration. Actually, a sequence of goal configurations is usually inferred from a high-level description of the task to be accomplished by the robot. For example, a person may request the robot to bring him/her a cup of coffee. Such a task requires (at least) achieving two goal configurations in sequence, one located near the coffee machine and the other close to the person. An important step toward generating a motion plan is to generate a path connecting the initial to the goal configuration. This does not mean that this path *is* the motion plan. Indeed, reaction to unexpected obstacles and imperfect low-level control may lead the robot to follow a slightly different path. However, it is an essential piece of information to be used by the rest of the navigation

system. For instance, it may be used to identify important landmarks and generate a sensory plan to monitor the execution of the motion.

In fact, because a generated free path may not be *the* motion plan to be executed, the navigation system may invoke the path planner multiple times with various constraints in order to obtain several possible paths. For instance, the navigation system may look for a free path that also allows the sensors to detect a sufficient number of environmental landmarks in order to facilitate the monitoring of the motion. Hence, the path planner is embedded in a larger system (the navigation system), requiring that it be efficient and reliable (i.e. well-behaved).

The configuration space of a mobile robot is typically a three-dimensional subspace of $\mathbf{R}^2 \times S^1$ (where S^1 denotes the unit circle). Two dimensions correspond to translational degrees of freedom; the third dimension corresponds to a rotational degree of freedom. In that context, we have developed a hierarchical path planner that generates abstraction problems by approximating the constraints defining the robot's free space. This planner incorporates the constraint representation and hierarchical search techniques introduced in Subsection 1.3.1. It is intended for a single mobile robot, and it does not use problem decomposition as the other means to generate abstraction problems. Our path planner has been used successfully as part of a contingency-tolerant robot navigation system [Choi, Zhu and Latombe, 1989] [Choi and Latombe, 1991] developed for the GOFER project in the Stanford Computer Science Robotics Laboratory [Caloud et al., 1990]. This navigation system takes advantage of the fact that our planner generates channels. These channels are directly used by the navigation system to keep some freedom of choice at execution time so as to avoid small unexpected obstacles.

Besides its application to robot navigation, our planner could also find application, say, in construction and maintenance planning of civil engineering facilities (e.g. power plants) where reasoning about accessibility is important. For example, in generating a maintenance plan for a nuclear power plant, one has to determine free paths for removing equipment for off-site repair. When no paths exist, one may hypothesize additional component removals and run the planner again, until collision-free paths

for the pieces of equipment to be repaired are found. Also, by running the path planner on multiple simulation problems during the design of a plant facility, one could produce a better design that facilitates both the construction and the future maintenance of the plant (concurrent engineering). Although path planning has seldom been applied to such domains so far (with a few exceptions such as the CAD system developed in Toshiba [Kondo and Ohtomi, 1991]), our planner has been designed to help solve the kind of problems mentioned above. In particular, it can plan the path of non-convex "robots".

Figure 1.1 shows paths generated by our planner for a variety of "robots" and workspaces.

1.4.2 Pipe Layout Design

The Pipe Layout Design (PLD) problem consists of finding the layout of pipes in a two-dimensional or a three-dimensional workspace. The pipes must connect terminals (or nozzles) of given locations and avoid collision with obstacles (machines, walls, structure) of given locations and shapes. The layout must often satisfy additional constraints related to the process carried out in the pipes (e.g. a water pipe should not be hanged above any electrical equipment) and to the design of mechanical support for the pipes (e.g. pipes should traverse designated structures providing support), as well as to the constructability and the ease of operation and maintenance of the pipe system designed (e.g. valves should be accessible). Furthermore, there are optimality criteria regarding the length and number of turns of the pipes which should be taken into account.

PLD is a problem of major significance in chemical, refinery, and power plant design [Gunn and Al-Asadi, 1987], and in ship and submarine design [Wangdahl, 1974] [Sheridan, 1976]. It is complex and time consuming due to the large number of pipes and constraints involved. It often requires many iterations before a satisfactory design can be obtained. Therefore, any progress toward automating PLD will benefit these industries in terms of higher quality, reduced cost, and shorter turn-around time. The PLD problem is well suited for exploring the problem decomposition approach
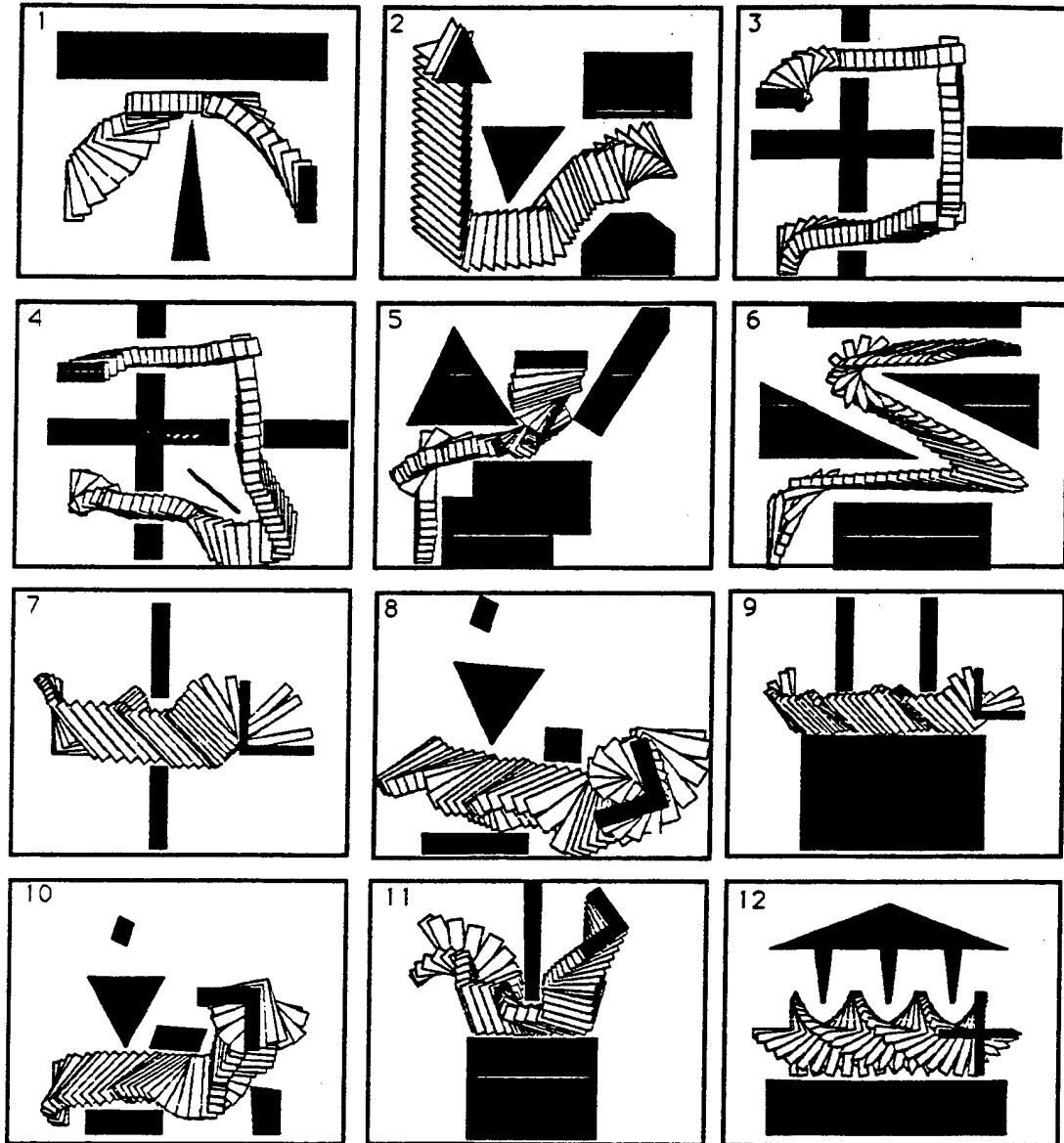
Figure 1.1: Various examples of motion planning problems

because the large number of pipes (from hundreds to thousands) involved in a pipe routing problem, in contrast to a few robots involved in a multiple robot navigation problem, provides a rich example for studying subproblem interactions.

We have developed a pipe router that applies an approximate cell decomposition method to generate pipe routes. Since there are usually many routes to generate, the configuration space of the overall problem has a high dimension. We cope with this difficulty by considering each pipe separately, hence decomposing the original problem into subproblems which have a lower-dimensional configuration space. Our pipe router makes use of the backtracking and lesser-commitment techniques introduced in Subsection 1.3.2 to deal with interaction among pipes. Figure 1.2 shows an example of a pipe layout generated by the implemented pipe router described in Chapter 4.

The basic pipe routing problem studied in this thesis is defined in Section 3.1 and is a simplification of the more complex pipe layout design problem described above. In this basic pipe routing problem, we are only concerned with connecting the terminals with pipes and we ignore other design constraints on the pipes (e.g. process constraints and structural constraints) that are important for most practical pipe layout design problems. [Zhu and Latombe, 1991b] describes a pipe layout design system that incorporates these other design constraints into the routing algorithms developed in this thesis.

The problem of pipe routing has been addressed previously in several pipe layout design systems. In the appendix, we provide a brief review of these systems. In a related domain, VLSI and PCB design, there has been a great deal of effort aimed at developing efficient routing systems. A review of VLSI routing is also given in the appendix.

1.5 Limitations

The advantage of the problem abstraction approach is intuitively obvious, and the experimental results obtained with the two implemented systems provide additional
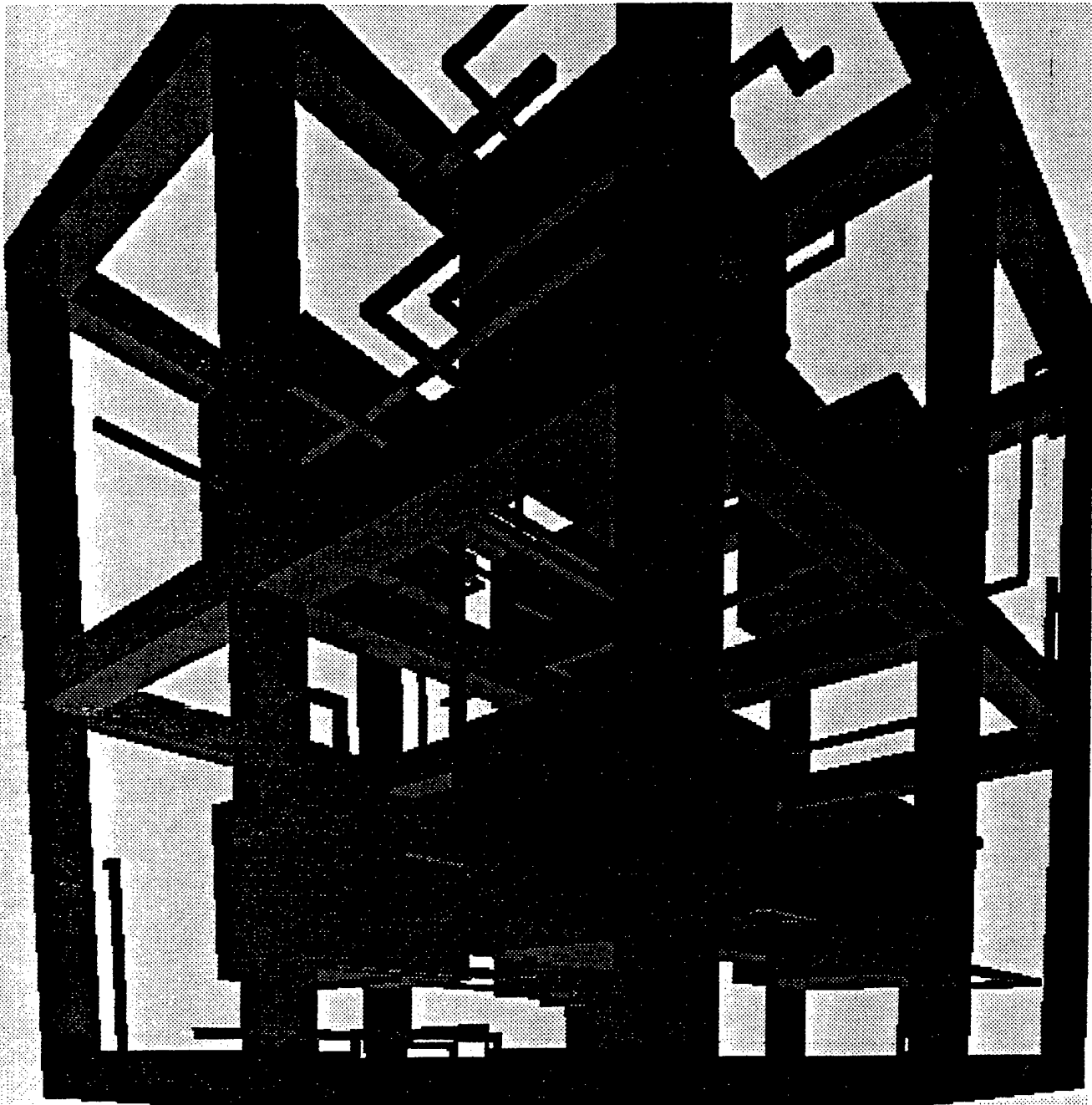
Figure 1.2: An example of the pipe routing problem

evidence of this value. Nevertheless, this approach and the techniques we have developed to implement it have some limitations that should be mentioned.

Constraint approximation as a way to generate abstraction problems can work well if the space is relatively uncluttered, so that a rather gross approximation of the obstacles still leaves enough free space for the robot to navigate through. If this is not the case, we may have to generate an excessive number of abstraction problems in order to approximate the obstacles accurately enough to find a solution. Hence, the cost of generating and solving these abstraction problems may be larger than the cost of solving the original problem.

Problem decomposition is efficient only if the original problem is decomposable into weakly interacting subproblems. If the interaction among the subproblems increases, the decomposition becomes artificial, and the number of backtracking operations may become impractical, even when lesser-commitment techniques are combined with selective backtracking algorithms.

These limitations may affect the well-behavedness of a path planner based on the problem abstraction approach. In general, if the solutions to the original problem are abundant, so that there may exist a problem abstraction that is much simpler than the original problem and it contains a non-empty subset of the solutions, then the problem approximation approach is very efficient. But as the set of solutions to the original problem decreases, the performance of the problem abstraction approach degrades accordingly. For a path planner based on the problem abstraction approach, one possible remedy would be to switch to an exact and complete method when it has failed to find a solution (or the absence of solution) after a pre-specified amount of time.

The above limitations are inherent to the problem abstraction approach. There are other limitations related to the specific algorithms that we have developed to implement this approach. For example, the choice of rectangloid cells used in the decomposition of the configuration space imposes a limitation on how accurately we can approximate the free space with a given number of cells; in sequential routing,

the backtracking algorithm treats each pipe as a unit and thus it sometimes fails to find a solution even though one exists; etc. These limitations also have impact on the behavior of the path planning systems. These limitations and their impact are discussed in detail in the chapters where the algorithms are described.

1.6 Summary and Outline

In this thesis, we address the problem of designing path planning algorithms that are both efficient and reliable. We propose a novel approach for solving path planning problems which consists of finding and solving an appropriate abstraction of the original problem by searching a space of possible abstractions. We argue that in order for this approach to be efficient, a tighter integration between geometric reasoning algorithms and search algorithms is critical.

This thesis is devoted to developing evidence to support this argument. In particular, this thesis explores two approaches for generating abstractions of path planning problems, the constraint approximation approach and the problem decomposition approach. The thesis focuses on the many issues raised by our attempt to develop algorithms that more tightly integrate geometry with search. The core of the thesis is organized into three chapters.

In chapter 2 we focus our study on the constraint approximation approach in the context of robot motion planning. In this chapter we develop a new method for approximate cell decomposition based on constraint reformulation techniques and an efficient hierarchical search algorithm with error discovery and recording mechanism. In chapters 3 and 4 we explore the problem decomposition approach in the context of pipe routing. In chapter 3 we describe a sequential routing algorithm with selective backtracking which can deal with weak subproblem interactions efficiently. In chapter 4 we present a parallel routing algorithm based on constraint propagation techniques which provides a more effective way for dealing with stronger subproblem interactions.

In each of these three chapters we discuss the various issues underlying the approach and the techniques which we have developed to address these issues. We

also describe the implementation of these techniques in a path planning system for a two-dimensional polygonal robot allowed to translate and rotate (in Chapter 2) and in a pipe routing system for routing multiple pipes in a three-dimensional space (in Chapters 3 and 4). We present experimental results obtained with these implemented systems.

The appendix contains a survey of related work on pipe routing and VLSI routing.

Chapter 2

Hierarchical Path Planning

In this chapter we explore the constraint approximation approach for generating problem abstractions in the context of a mobile robot path planning system. The robot is modeled as a planar rigid object that can translate and rotate freely in a two-dimensional workspace. The path planning method used is the approximate cell decomposition method. A problem abstraction is generated by approximating the constraints defining the robot's free space. The two key issues addressed in this chapter are cell decomposition and search. With respect to cell decomposition, we have developed a set of new algorithms based on "constraint reformulation" techniques that result in a more efficient decomposition of the robot's free space. With respect to search, we propose new hierarchical search algorithms that take advantage of previous, unsuccessful search effort. These algorithms are implemented in a path planner. Based on our experiments on a variety of examples and taking into account the relative speed of the computers, this planner is significantly (approximately 10 times) faster than previous planners based on the same approximate cell decomposition approach.

This chapter is organized as follows. In Section 2.1 we specify the problem and give a detailed overview of the approximate cell decomposition approach. In Sections 2.2 and 2.3 we investigate the cell decomposition and graph searching problems, respectively, and we describe in detail the new algorithms which we developed. In Section 2.4 we present some of the experiments that we conducted with the implemented planner. In Section 2.5 we discuss the limitations of the algorithms developed in this

chapter and suggest possible remedies. In Section 2.6 we summarize the results.

2.1 Problem and Approach

2.1.1 The Path Planning Problem

The path planning problem discussed in this chapter consists of finding a collision-free continuous path for a planar rigid robot $\mathcal{A}$ between an initial configuration (position and orientation) q_{init} and a goal configuration q_{goal}. The robot is allowed to translate and rotate in a two-dimensional workspace $\mathcal{W}$, which is populated by a set of obstacles, $\mathcal{B}_i$ ($i = 1, ..., n$). Both the robot and the obstacles are polygons.
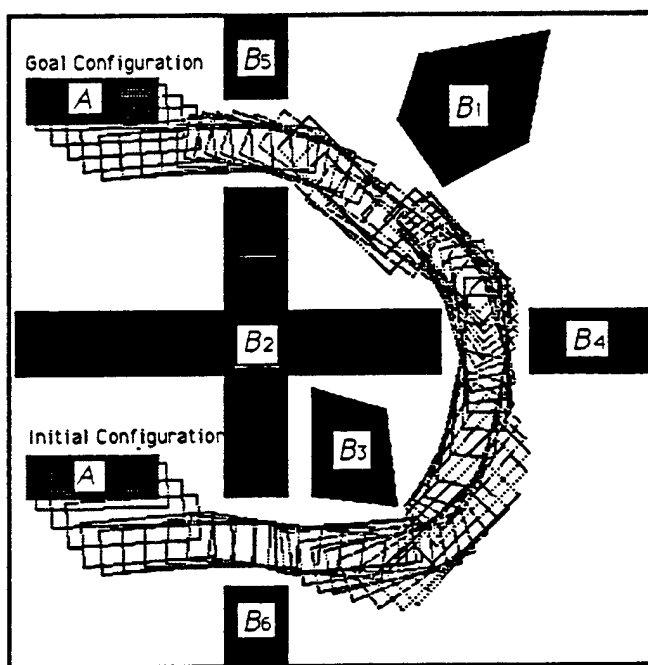
Figure 2.1: An example of a path planning problem

We formulate the path planning problem as follows:

Given the geometry of $\mathcal{A}$ and the $\mathcal{B}_i$'s, the locations of the $\mathcal{B}_i$'s, and q_{init} and q_{goal} of $\mathcal{A}$, find a continuous path connecting q_{init} to q_{goal} while

avoiding collision between $\mathcal{A}$ and the $\mathcal{B}_i$'s.

Figure 2.1 illustrates a typical example of this class of path planning problems, together with a solution path.

2.1.2   Notion of Configuration Space

A **configuration** of $\mathcal{A}$ is a specification of the position of every point of $\mathcal{A}$ relative to a fixed coordinate system embedded in the workspace $\mathcal{W}$. If $\mathcal{A}$ is a single planar rigid object that is allowed to translate and rotate, its configuration can be specified by the triplet (x, y, θ), where x and y are the coordinates of a **reference point** $O_{\mathcal{A}}$ chosen in $\mathcal{A}$ and θ is an angle modulo 2π defining the orientation of $\mathcal{A}$. The **configuration space** (or C-space) $\mathcal{C}$ of $\mathcal{A}$ is the set of all of its configurations. The subset of $\mathcal{W}$ occupied by $\mathcal{A}$ at configuration $\mathbf{q}$ is denoted by $\mathcal{A}(\mathbf{q})$.

Each obstacle $\mathcal{B}_i$ maps in $\mathcal{C}$ as a closed region denoted by $\mathcal{CB}_i$ and called **C-obstacles**. This region is defined by:

$$\mathcal{CB}_i = \{\mathbf{q} \in \mathcal{C} \mid \mathcal{A}(\mathbf{q}) \cap \mathcal{B}_i \neq \emptyset\}.$$

The region:

$$\mathcal{C}_{free} = \mathcal{C} - \bigcup_{i \in [1,n]} \mathcal{CB}_i$$

is called the **free space**. A **collision-free path** (more simply, a **free path**) is any continuous map $\tau : [0, 1] \rightarrow \mathcal{C}_{free}$.

$\mathcal{CB}_i$ is a three-dimensional volume in $\mathcal{C}$ (Figure 2.2) whose cross-section at any orientation θ is defined by [Lozano-Pérez, 1983]:

$$\mathcal{CB}_i(\theta) = \mathcal{B}_i \ominus \mathcal{A}(0, 0, \theta)$$

where $\mathcal{A}(0, 0, \theta)$ denotes the region occupied by $\mathcal{A}$ at configuration $(0, 0, \theta)$ and $\ominus$ is the symbol for the Minkowski difference ($A \ominus B = \{a - b \mid a \in A, b \in B\}$). Since $\mathcal{A}$ and $\mathcal{B}_i$ are polygons, $\mathcal{CB}_i(\theta)$ is also a polygon. Each edge of this polygon is the locus of $O_{\mathcal{A}}$ when $\mathcal{A}$ translates at a fixed orientation θ in such a way that an edge

(resp. a vertex) of $\mathcal{A}$ stays in contact with a vertex (resp. an edge) of $\mathcal{B}_i$ (see Figure 2.2). A contact between an edge of $\mathcal{A}$ and a vertex of $\mathcal{B}_i$ is called a **Type A** contact. When $\mathcal{A}$ rotates slightly, the corresponding edge of $\mathcal{CB}_i$ rotates by the same angle. A contact between a vertex of $\mathcal{A}$ and an edge of $\mathcal{B}_i$ is called a **Type B** contact. When $\mathcal{A}$ rotates slightly, the corresponding edge of $\mathcal{CB}_i$ translates.
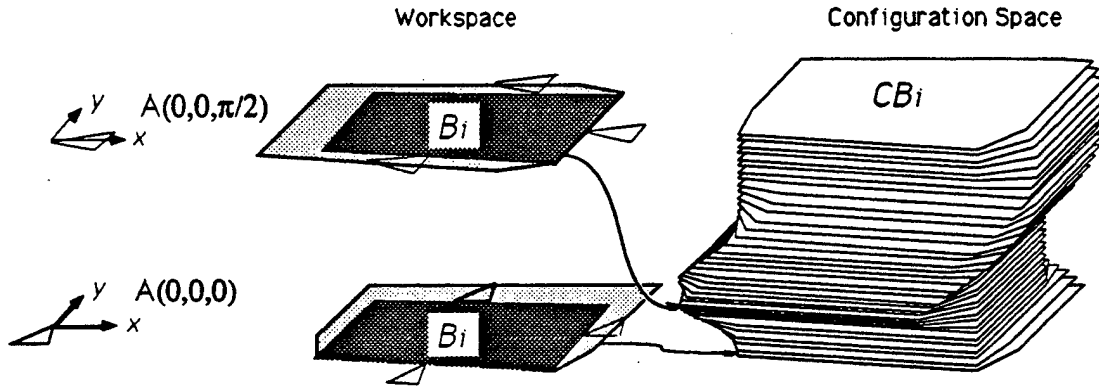
Figure 2.2: A C-Obstacle

Therefore, the C-obstacle $\mathcal{CB}_i$ is a three-dimensional volume bounded by patches of ruled surfaces, which we call **C-facets**. Each C-facet is generated by a straight line segment of variable length, which remains parallel to the xy plane and which translates and/or rotates. A C-facet created by a contact of Type A (resp. Type B) is called a Type A (resp. Type B) C-facet. Each C-facet is comprised between two limit orientations, beyond which the contact that creates the C-facet is no longer feasible.

The geometry of $\mathcal{CB}_i$, when $\mathcal{A}$ and $\mathcal{B}$ are polygons, is studied in depth in various publications, e.g. [Lozano-Pérez, 1983] [Donald, 1984] [Avnaim and Boissonnat, 1988] [Brost, 1989]. If $\mathcal{A}$ and $\mathcal{B}_i$ are both convex polygons, $\mathcal{CB}_i$ is bounded by $O(n_{\mathcal{A}}n_B)$ C-facets, where $n_{\mathcal{A}}$ and n_B are the number of edges of $\mathcal{A}$ and $\mathcal{B}$, respectively. If $\mathcal{A}$ and $\mathcal{B}$ are non-convex, $\mathcal{CB}$ has $\Omega(n_{\mathcal{A}}^3 n_B^3)$ C-facets in the worst case [Avnaim and Boissonnat, 1988].

2.1.3 Rectangloid Decomposition

In the following, we will assume, without practical loss of generality, that the range of possible values for x and y (the coordinates of O_A) are closed intervals $[x_{min}, x_{max}]$ and $[y_{min}, y_{max}]$. We represent C as a closed rectangloid

$$\mathcal{K} = [x_{min}, x_{max}] \times [y_{min}, y_{max}] \times [0, 2\pi] \subset \mathbf{R}^3$$

with the two cross-sections at $\theta = 0$ and $\theta = 2\pi$ procedurally identified.

Let κ be a rectangloid of the form:

$$[x_1, x_2] \times [y_1, y_2] \times [\theta_1, \theta_2] \subseteq \mathcal{K}.$$

We define a **rectangloid decomposition** of κ as a collection of rectangloids, $\{\kappa_j\}_{j=1,\dots,n}$, such that:

1. κ is equal to the union of the κ_j, i.e.:

$$\kappa = \bigcup_{j=1}^{n} \kappa_j.$$

2. The κ_j are non-overlapping:[1]

$$\forall j_1, j_2 \in [1, n], \ j_1 \neq j_2 : \ int(\kappa_{j_1}) \cap int(\kappa_{j_2}) = \emptyset.$$

Each rectangloid κ_j is called a **cell** of κ in the decomposition. Two cells κ_{j_1} and κ_{j_2} are **adjacent** iff their intersection is not a set of measure zero in $\mathbf{R}^2$, i.e. the intersection of two of their faces is a surface of non-zero area. (The intersection is computed by taking into account that the cross-sections at 0 and 2π are identified.)

A cell κ_j is classified as:

- EMPTY, iff its interior $int(\kappa_j)$ intersects no C-obstacle, i.e. $int(\kappa_j) \cap \bigcup_i \mathcal{CB}_i = \emptyset$;

- FULL, iff κ_j is entirely contained in the union of the C-obstacles, i.e. $\kappa_j \subset \bigcup_i \mathcal{CB}_i$;

---

[1]Throughout this thesis, we say that two closed sets are non-overlapping iff their interiors do not intersect.

- MIXED, otherwise.

Given an initial configuration

$$\mathbf{q}_{init} = (x_{init}, y_{init}, \theta_{init})$$

and a goal configuration

$$\mathbf{q}_{goal} = (x_{goal}, y_{goal}, \theta_{goal}),$$

a sequence of EMPTY cells κ_k, where $k = 1, ..., p$ in a rectangloid decomposition of $\mathcal{K}$ is called an EMPTY **channel** iff it satisfies the following three properties:

- $(x_{init}, y_{init}, \theta_{init}) \in \kappa_1$;

- $(x_{goal}, y_{goal}, \theta_{goal}) \in \kappa_p$;

- $\forall k \in [1, p - 1] : \kappa_k$ and κ_{k+1} are adjacent.

Let $(\kappa_k)_{k=1,...,p}$ be an EMPTY channel. Any path connecting $(x_{init}, y_{init}, \theta_{init})$ to $(x_{goal}, y_{goal}, \theta_{goal})$ and lying entirely in the interior of $\bigcup_{k=1}^{p} \kappa_k$ is a free path. Such a path can easily be constructed as an open polygonal line. If necessary, it may also be smoothed by fitting spline curves [Kant and Zucker, 1986a].

A sequence of EMPTY and MIXED cells κ_k, $k = 1, ..., p$, is called a MIXED **channel**, iff it has the same three properties as stated above. A MIXED channel may contain a free path connecting the initial to the goal configuration, but there is no guarantee that this is the case.

2.1.4 Hierarchical Path Planning

Hierarchical path planning attempts to generate an EMPTY channel by iteratively applying the following two steps: decomposition and search. Therefore it constructs successive rectangloid decompositions of $\mathcal{K}$ and searches through the sets of cells until an EMPTY channel has been extracted.

Let $\mathcal{P}_l$, $l = 0, 1, ...$, denote the successive decompositions of $\mathcal{K}$, with $\mathcal{P}_0 = \{\mathcal{K}\}$. Each decomposition $\mathcal{P}_l$, $l > 0$, is obtained from the previous one, $\mathcal{P}_{l-1}$, by decomposing one or several MIXED cells, the other cells being unchanged.

Whenever a decomposition $\mathcal{P}_l$, $l \geq 0$, is generated and its cells are labelled EMPTY, FULL, and MIXED, an undirected graph denoted by CG_l is constructed:

- The nodes of CG_l are the EMPTY and MIXED cells in $\mathcal{P}_l$.

- Any two nodes are connected by a link iff they are adjacent.

This graph is called the **cell-connectivity graph, or cg**, of the decomposition $\mathcal{P}_l$.

Once constructed[2], the graph CG_l is searched for an EMPTY or MIXED channel. Three outcomes are possible:

1. An EMPTY channel is found. Then, the planner returns success.

2. No EMPTY or MIXED channel is found. Then, the planner returns failure.

3. A MIXED channel is found, but not an EMPTY channel.

In the third case, the planner proceeds recursively by decomposing the MIXED cells contained in the MIXED channel. Hence, the planner iteratively refines the "interesting" areas.

Let us assume that the union of the C-obstacles forms a manifold with boundary, i.e., every point of this manifold admits a neighborhood that is topologically equivalent to either an open ball or a half plane. This assumption allows us to preclude the pathological cases shown in Figure 2.3 where two C-obstacle regions are connected by a point, a line, or a plane. Under this assumption the planning process sketched above can be made complete – i.e., guaranteed to terminate successfully, whenever a free path exists, and to return failure, otherwise. This requires that some simple details of the algorithms be worked out appropriately. However, for an unknown problem, there is no time bound on the process, since there is no lower bound on the size of the cells which may have to be generated.

The worst-case time complexity can be bounded at the expense of completeness,

---

[2]Actually, the graph is a by-product of the decomposition and is constructed concurrently with the decomposition.
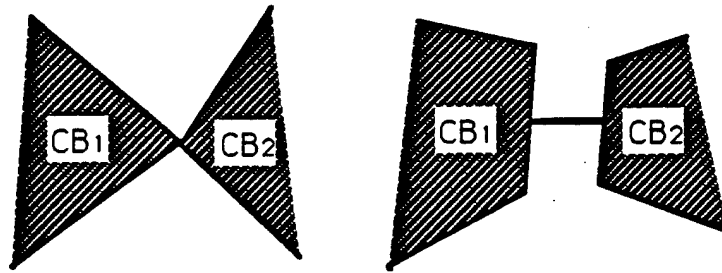
Figure 2.3: Pathological cases

by imposing constraints on the decomposition of any MIXED cell κ. One possible constraint is that the total volume of the EMPTY and FULL cells extracted from κ be greater than a predefined ratio of the volume of κ; in addition, every MIXED cell resulting from the decomposition of κ which has any of its dimensions (side length) smaller than a predefined value is relabelled FULL. Another possible constraint is that every generated cell should have its dimensions greater than pre-specified values. The choice of the constraints on the decomposition of a MIXED cell typically depends on the decomposition technique. If the decomposition algorithm fails to find a decomposition of κ satisfying the constraint, κ is said to be non-decomposable and is re-labelled FULL.

If constraints are imposed on MIXED cell decomposition, it is no longer guaranteed that a free path is found, whenever one such path exists. However, the planner can be made "resolution-complete", i.e. if an EMPTY channel exists at the resolution determined by the constraints, it will be found.

If the C-obstacle region is not a manifold with boundary, then, in the absence of constraints on the decomposition of MIXED cells, the planning process may loop forever. The constraints given above provide a way to guarantee the termination of the planning process in bounded time.

In the presence of uncertainty in robot control, a minimal size requirement may also have to be imposed both on the cells in a channel and on the intersection of two successive cells in the channel, in order to allow the robot to move safely despite control error.

2.1.5 Issues

Despite the conceptual simplicity of the two main steps of the above approach (cell decomposition and graph searching), their efficient implementation raises delicate questions not thoroughly addressed in previous work. In our research, we found that efficiency can be drastically increased by shifting from naive answers to these questions to more sophisticated ones.

The issue raised by cell decomposition is how to maximize the volume of the EMPTY and FULL cells resulting from the decomposition of a MIXED cell, in order to make it possible to find a path (or the absence of a path) as quickly as possible. In particular, the classical blind 2^m-tree (e.g., quadtree, octree) decomposition technique mentioned in Chapter 1 has the drawback of decomposing many MIXED cells into smaller MIXED cells. We propose below a new "constraint reformulation" technique, which provides much better experimental results than earlier decomposition techniques. It consists of approximating the obstacles intersecting with the MIXED cell to be decomposed with a collection of rectangloids, and computing the complement of these rectangloids in the cell. Two types of approximation are used, "bounding" and "bounded" approximations. The first is used to produce EMPTY cells, the second to produce FULL cells. This technique is discussed in Section 2.2.

The issue raised by graph searching is how to take advantage of unsuccessful search work done at prior levels of approximation in order to guide the current level of search, since most of the search graph remains the same from one level to the next (only the portions of the graph which correspond to the newly decomposed MIXED cells are modified). We propose a set of search techniques based on appropriate representation of the search graph and on the recording of failure conditions. These techniques, inspired from those implemented in dependency-directed backtracking systems [Stallman and Sussman, 1977] [Latombe, 1979], help the path planner to avoid repeating the same mistakes several times. These hierarchical search techniques are discussed in Section 2.3.

2.2   Cell Decomposition

2.2.1   Issue

The issue considered in this section is that of generating a "good" rectangloid decomposition $\{\kappa_j\}_{j=1,\ldots,n}$ of a given MIXED cell κ. For the efficiency of the planning process, the generated decomposition should simultaneously satisfy the following two goals:

- The number of cells in the decomposition should be reasonably small, in order to keep the size of the search graph as small as possible.

- The volume of the EMPTY and FULL cells should be large relative to the total volume of κ.

These two goals may conflict since one obvious way to achieve the second goal is to produce many small cells, which contradicts with the first one. An empirical quantitative measure of the efficiency of the decomposition of a cell may be:

$$E = \frac{1}{N_{EMPTY} + N_{MIXED}} \frac{V_{EMPTY} + V_{FULL}}{V_\kappa}$$

where:

- N_{EMPTY} (resp. N_{MIXED}) is the number of newly generated cells labelled EMPTY (resp. MIXED),

- V_{EMPTY} (resp. V_{FULL}) is the total volume of newly generated EMPTY (resp. FULL) cells,

- V_κ is the volume of the cell κ.

A greater E corresponds to a better decomposition. The computational cost of generating a decomposition should also remain as small as possible.

2.2.2 Previous Approaches

One simple method for decomposing a MIXED cell is to partition it into 2^m cells of equal dimensions, where m is the dimension of the configuration space . The decomposition obtained with this method is called a "quadtree", when $m = 2$, and an "octree", when $m = 3$ [Ayala et al., 1985]. The application of this method is reported in [Kambhampati and Davis, 1986] (quadtree) and [Faverjon, 1984] (octree). The advantage of this method is that it yields a decomposition easily representable in a tree structure of degree 2^m. Its drawback is that cells are produced blindly; most of the time none or a few of the newly generated cells are EMPTY or FULL. Thus it tends to produce a huge number of cells. We call such a method a "decompose-and-label" method because it produces cells in a first phase of processing and labels these cells in a second phase.

Another method is described in [Brooks and Lozano-Pérez, 1985]. The basic idea is to consider potential cuts of the cell, score them, and choose the one with the best score. The potential cuts are chosen wherever a C-surface[3] will go through a vertex of one of the new cells generated by the decomposition. The scoring function favors cuts which do not generate small cells, i.e. the cuts that are closer to the mid-point of an edge are preferred. The scoring function also attempts to minimize the number of C-surfaces intersected by each new cell, in order to reduce future computations. This method has the drawback of treating each C-surface individually and then combining the effect of the various C-surfaces in a global scoring function. Although certainly better than a 2^m-tree, the resulting decomposition may still be far from optimal. It may also incorrectly label a cell that intersects no C-obstacle as MIXED[4].

In [Lozano-Pérez, 1983], Lozano-Pérez describes a method which consists of slicing the orientation axis into intervals, computing the area swept out by $\mathcal{A}$ when it rotates about its reference point in each interval, approximating the swept area as a polygon, and growing the obstacles by this polygon. The result is a decomposition

---

[3]A *C-surface* is the infinite ruled surface that support a C-facet.

[4]This conservative "error" gets corrected when the decomposition proceeds deeper [Latombe, 1991].

of configuration space into prismatic cells, which are either empty or not empty (the latter are not characterized further). A path is searched among the empty cells only. If it fails, the angular intervals are refined. The "swept-area" method proposed below has some similarity with this approach. Our method, however, generates EMPTY, FULL and MIXED cells.

2.2.3 Constraint Reformulation

We propose a different approach for decomposing MIXED cells. It consists of first approximating each C-obstacle lying in the cell κ to be decomposed as a collection of non-overlapping rectangloids. The complement of a union of rectangloids within a rectangloid region is also a union of rectangloids, which can be computed easily. This yields a rectangloid decomposition of κ. We call this approach *constraint reformulation*, since it basically consists of reformulating the constraints imposed by the C-obstacles into a form directly compatible with the format of the decomposition of $\mathcal{K}$ (rectangloid).

Let $\kappa = [x_1, x_2] \times [y_1, y_2] \times [\theta_1, \theta_2]$ be the MIXED cell to be decomposed. For every $i \in [1, n]$, we denote by $\mathcal{CB}_i[\kappa]$ the portion of $\mathcal{CB}_i$ contained in κ, i.e.:

$$\mathcal{CB}_i[\kappa] = \mathcal{CB}_i \cap \kappa.$$

Our planner generates and uses two types of approximation of C-obstacles, bounding and bounded approximations:

1. A **bounding approximation** of $\mathcal{CB}_i[\kappa]$ is a collection of non-overlapping rectangloids $\mathcal{R}_{ik}$'s, $k = 1, ..., p$, with $\forall k \in [1, p] : \mathcal{R}_{ik} \subset \kappa$ and $\mathcal{CB}_i[\kappa] \subseteq \bigcup_{k=1,...,p} \mathcal{R}_{ik}$.

2. A **bounded approximation** of $\mathcal{CB}_i[\kappa]$ is a collection of non-overlapping rectangloids $\mathcal{R}'_{ik}$'s, $k = 1, ..., p'$, with $\bigcup_{k=1,...,p'} \mathcal{R}'_{ik} \subseteq \mathcal{CB}_i[\kappa]$.

The EMPTY cells of the decomposition of κ are obtained by computing the complement of $\bigcup_i \bigcup_k \mathcal{R}_{ik}$ in κ. The FULL cells are the $\mathcal{R}'_{ik}$'s. The MIXED cells are obtained by computing the complement of $\bigcup_i \bigcup_k \mathcal{R}'_{ik}$ in every $\mathcal{R}_{ik}$.
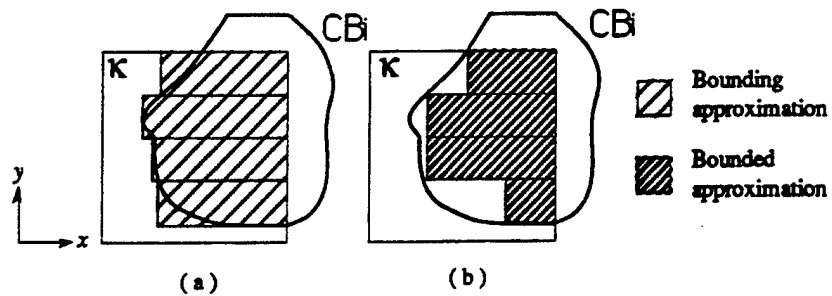
Figure 2.4: Bounding and Bounded Approximations
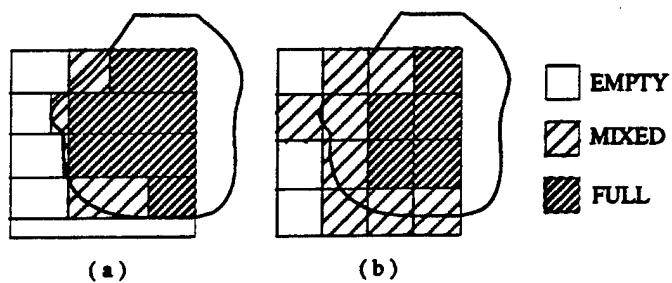
Figure 2.5: Approximate cell decomposition using constraint reformulation method (a) vs. quadtree method (b)

Figure 2.4 illustrates the notion of bounding and bounded approximations in a two-dimensional space. Figure 2.5a illustrates the rectangloid decomposition of a cell κ into EMPTY, FULL and MIXED cells, using these two approximations. Compared with the quadtree decomposition at a similar resolution as shown in Figure 2.5b, we see that the decomposition using bounding and bounded approximations produces much a smaller MIXED area, larger EMPTY and FULL areas, and a smaller number of cells.

2.2.4 Outline of the Algorithm

There are arbitrarily many ways to generate bounding and bounded approximations of a C-obstacle CB in a cell κ. Our method first computes "outer" and "inner" projections of CB on the xy plane and next on the x or y axis. Then, it lifts back the projection into rectangloid cells. It consists of the following two steps:

1. **Projection on the xy plane:** The $[\theta_1, \theta_2]$ interval of a MIXED cell κ is cut into non-overlapping subintervals $[\gamma_u, \gamma_{u+1}]$, $u = 1, ..., r$, $r \geq 1$, with $\gamma_1 = \theta_1$ and $\gamma_{r+1} = \theta_2$. We denote by κ^u the rectangloid $[x_1, x_2] \times [y_1, y_2] \times [\gamma_u, \gamma_{u+1}]$.

   For every $u \in [1, r]$, we compute the **outer projection** and the **inner projection** of $CB[\kappa^u]$ into the xy plane. These two projections, denoted by $OCB_{xy}[\kappa^u]$ and $ICB_{xy}[\kappa^u]$ respectively, are defined as:

   $$OCB_{xy}[\kappa^u] = \{(x,y) \, / \, \exists \theta \in [\gamma_u, \gamma_{u+1}] : (x, y, \theta) \in CB[\kappa^u],$$

   $$ICB_{xy}[\kappa^u] = \{(x,y) \, / \, \forall \theta \in [\gamma_u, \gamma_{u+1}] : (x, y, \theta) \in CB[\kappa^u].$$

   Clearly, we have: $ICB_{xy}[\kappa^u] \subseteq OCB_{xy}[\kappa^u]$.

2. **Projection on the x or the y axis:** For every $u \in [1, r]$, the interval $[x_1, x_2]$ or $[y_1, y_2]$, whichever is longer, is cut into non-overlapping subintervals.

   Let us assume that $[x_1, x_2]$ is subdivided (a similar presentation would be made in the case where $[y_1, y_2]$ was subdivided). The generated subintervals are $[a_v, a_{v+1}]$,

$v = 1, ..., s$, $s \geq 1$, with $a_1 = x_1$ and $a_{s+1} = x_2$. We let κ^{uv} denote the rectangloid $[a_v, a_{v+1}] \times [y_1, y_2] \times [\gamma_u, \gamma_{u+1}]$.

For every $v \in [1, s]$, we compute the outer projection of $\mathcal{OCB}_{xy}[\kappa^u]$ on the y axis, i.e.:

$$\mathcal{OCB}_y[\kappa^{uv}] = \{y \ / \ \exists x \in [a_v, a_{v+1}] : (x, y) \in \mathcal{OCB}_{xy}[\kappa^u]\}$$

and the inner projection of $\mathcal{ICB}_{xy}[\kappa^u]$ on the y axis, i.e.:

$$\mathcal{ICB}_y[\kappa^{uv}] = \{y \ / \ \forall x \in [a_v, a_{v+1}] : (x, y) \in \mathcal{ICB}_{xy}[\kappa^u]\}.$$

Both $\mathcal{OCB}_y[\kappa^{uv}]$ and $\mathcal{ICB}_y[\kappa^{uv}]$ are sets of intervals where

$$\mathcal{ICB}_y[\kappa^{uv}] \subseteq \mathcal{OCB}_y[\kappa^{uv}].$$

Each rectangloid $[a_v, a_{v+1}] \times [b, b'] \times [\gamma_u, \gamma_{u+1}]$, where $[b, b'] \in \mathcal{OCB}_y[\kappa^{uv}]$ (resp. $\mathcal{ICB}_y[\kappa^{uv}]$), is a rectangloid $\mathcal{R}_{ik}$ (resp. $\mathcal{R}'_{ik}$) in the bounding (resp. bounded) approximation of $\mathcal{CB}_i[\kappa]$ (see Subsection 2.2.3).

The choice of the γ_u's and the a_v's is empirical. Various heuristics can be used, but most of them seem to have a limited effect on the average efficiency of the method. The only useful heuristic guideline is to keep the three dimensions of every MIXED cell approximately "homogeneous". Let δx, δy and $\delta \theta$ be these dimensions. We say they are "homogeneous" iff $\delta x \approx \delta y \approx \rho \delta \theta$, where ρ is the maximal distance between $O_{\mathcal{A}}$ and the points in the boundary of $\mathcal{A}$.

In the next two subsections, we describe the computation of the outer and inner projections of $\mathcal{CB}[\kappa^u]$ on the xy plane. We propose two different methods. The first, called "projection" method, consists of computing the projection of the surface of $\mathcal{CB}$ comprised between γ_u and γ_{u+1}, and clipping the subset of the projection contained in $[x_1, x_2] \times [y_1, y_2]$. The second method, called "swept-area" method, consists of computing the "outer" and the "inner" swept areas of $\mathcal{A}$ when it rotates around the reference point from orientation γ_u to orientation γ_{u+1}, and growing the obstacle $\mathcal{B}$ by these two areas. The projection method is preferred when the interval $[\gamma_u, \gamma_{u+1}]$ is small. The swept-area method runs significantly faster when this interval is large;

however, it does not exactly compute $\mathcal{ICB}_{xy}[\kappa^u]$, but a subset of it. The computation of $\mathcal{OCB}_y[\kappa^{uv}]$ and $\mathcal{ICB}_y[\kappa^{uv}]$ is quite simple and is not described here.

In the following, we assume that the reference point $O_{\mathcal{A}}$ is chosen in the interior of $\mathcal{A}$. In fact, the choice of $O_{\mathcal{A}}$ has some impact on the efficiency of the decomposition. The "best" location of $O_{\mathcal{A}}$ should minimize the area of the outer projection of $\mathcal{CB}[\kappa^u]$ on the xy plane, while maximizing the area of the inner projection, so that less MIXED cells are ultimately generated. The center of the "smallest enclosing circle" of $\mathcal{A}$ is probably close to minimizing the area of the outer projection, while the center of the "largest enclosed circle" of $\mathcal{A}$ should be close to maximizing the area of the inner projection. A compromise between these two locations is in general necessary, since they do not coincide. [Pignon et al., 1991] describes a method for computing the "best" reference point for a polygonal robot.

2.2.5 Projection Method

Principle. Let us first assume that both $\mathcal{A}$ and $\mathcal{B}$ are convex polygons. In $\mathbf{R}^2 \times [0, 2\pi]$, $\mathcal{CB}$ is a volume without hole, which is bounded by C-facets (see Subsection 2.1.2). Consider the point (x_0, y_0) in the xy plane and the segment $\{(x_0, y_0, \theta) \mid \theta \in [\gamma_u, \gamma_{u+1}]\}$ above this point. If the segment pierces a C-facet, then (x_0, y_0) is in $\mathcal{OCB}_{xy}[\kappa^u]$, but not in $\mathcal{ICB}_{xy}[\kappa^u]$. If it pierces no C-facet, then either (x_0, y_0) is not in $\mathcal{OCB}_{xy}[\kappa^u]$, or it is in $\mathcal{ICB}_{xy}[\kappa^u]$. The segment $\{(x_0, y_0, \theta) \mid \theta \in [\gamma_u, \gamma_{u+1}]\}$ pierces a C-facet e iff (x_0, y_0) lies in the projection of e in the xy plane.

We show below that each C-facet comprised between γ_u and γ_{u+1} projects on the xy plane according to a generalized polygon[5]. Hence, the projection of the boundary of $\mathcal{CB}$ that is comprised between γ_u and γ_{u+1} is the union of generalized polygons, each being the projection of a C-facet. This union is a "donut" shaped region (see Figure 2.6), with an outer boundary Γ_1 and an inner boundary Γ_2. The compact region bounded by Γ_1 is $\mathcal{OCB}_{xy}[\kappa^u]$. The compact region bounded by Γ_2 is $\mathcal{ICB}_{xy}[\kappa^u]$.

---

[5]A *generalized polygon* is a compact two-dimensional region bounded by a simple curve consisting of straight segments and circular arcs.
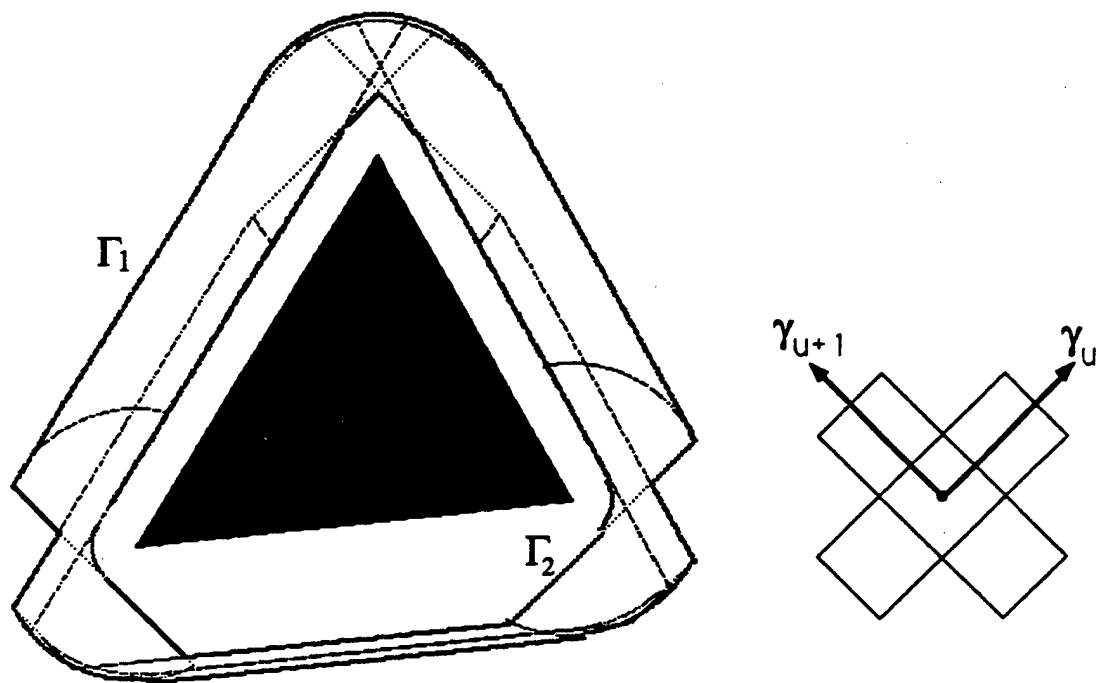
Figure 2.6: The projection on the xy plane of the C-patches comprised in an angular slice $[\gamma_u, \gamma_{u+1}]$

Therefore, the projection method consists of: (1) projecting all the C-facets (to the extent they are contained in the interval $[\gamma_u, \gamma_{u+1}]$) on the xy plane; (2) clipping the union of the projections by the rectangle $[x_1, x_2] \times [y_1, y_2]$ and, within this rectangle, tracking Γ_1 and Γ_2.

The speed of the projection method can be improved by associating with each interval $[\gamma_u, \gamma_{u+1}]$ the list of all the C-obstacles having a non-empty intersection with the interval, and for each of these C-obstacles the list of the C-facets that intersect the interval. If the interval gets decomposed further, only these C-facets have to be considered.

The projection method is efficient when the number of C-facets which have to be projected is reasonably small, that is when the interval $[\gamma_u, \gamma_{u+1}]$ is small.
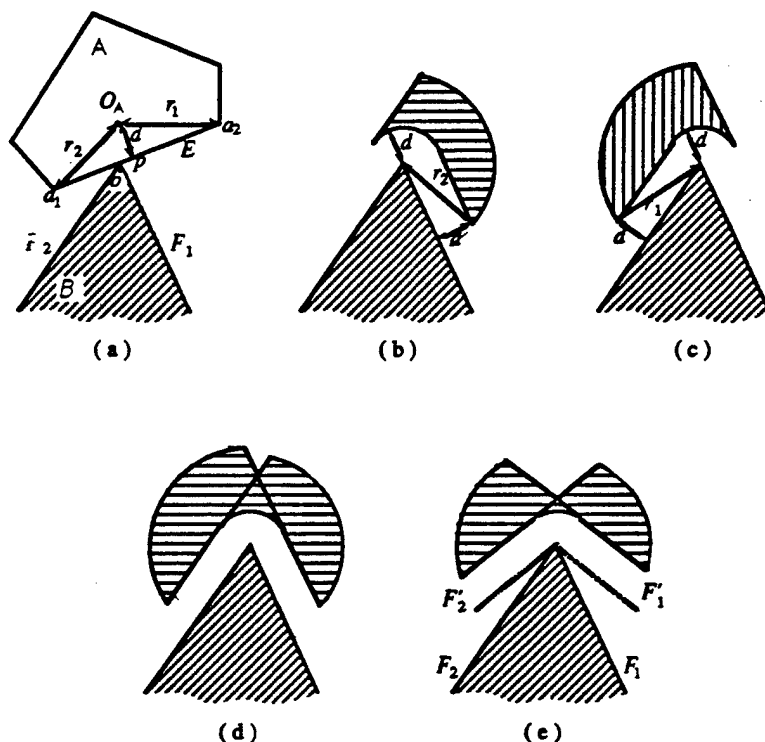
Figure 2.7: Projection of a Type A C-Facet

Projection of Type A C-Facet. Let e_A be a C-facet of Type A comprised between the limit orientations ϕ_1 and ϕ_2. The contact that generates e_A is illustrated in Figure 2.7a. It occurs between an edge E of $\mathcal{A}$ and a vertex b of $\mathcal{B}$. E connects two vertices of $\mathcal{A}$, a_1 and a_2. b is the extremity of two edges of $\mathcal{B}$, F_1 and F_2. $O_{\mathcal{A}}$ projects on the supporting line of E at the point p. We assume below that p is located between a_1 and a_2. (The case where p is outside the segment $\overline{a_1 a_2}$ is treated in a very similar fashion.) The orientation ϕ_1 (resp. ϕ_2) is achieved when the edge E is aligned with the edge F_1 (resp. F_2). Assuming that $[\phi_1, \phi_2] \subseteq [\gamma_u, \gamma_{u+1}]$, the projection of e_A on the xy plane is shown in Figure 2.7d. It is obtained as the union of two regions shown in Figures 2.7b and 2.7c.

The region in Figure 2.7b is the locus of $O_{\mathcal{A}}$, when $\mathcal{A}$ translates and rotates while the edge segment $\overline{a_1 p}$ stays in contact with the vertex b. This region is bounded by two circular arcs and two straight segments. The two arcs are centered at b. The smaller one is the locus of $O_{\mathcal{A}}$ when p coincides with b and $\mathcal{A}$ rotates from ϕ_1 to ϕ_2. The larger arc is the locus of $O_{\mathcal{A}}$ when a_1 coincides with b and $\mathcal{A}$ rotates from ϕ_1 to ϕ_2. The straight segments are the loci of $O_{\mathcal{A}}$ when $\mathcal{A}$ translates at fixed orientations ϕ_1 and ϕ_2 from the position where p and b coincide to the position where a_1 and b coincide. The region in Figure 2.7c is the locus of $O_{\mathcal{A}}$, when $\mathcal{A}$ translates and rotates while the edge segment $\overline{p a_2}$ stays in contact with the vertex b.

The case where $[\gamma_u, \gamma_{u+1}] \subset [\phi_1, \phi_2]$ can be treated in the same way, by drawing fictitious edges F_1' and F_2' from vertex b (see Figure 2.7e), so that, when E is aligned with F_1' (resp. F_2'), $\mathcal{A}$'s orientation is γ_u (resp. γ_{u+1}).

Projection of a Type B C-Facet. Let e_B be a C-facet of Type B comprised between the limit orientations ϕ_1 and ϕ_2. The contact that generates e_B is illustrated in Figure 2.8a. It occurs between a vertex a of $\mathcal{A}$ and an edge F of $\mathcal{B}$. a is the extremity of two edges of $\mathcal{A}$, E_1 and E_2. F connects two vertices of $\mathcal{B}$, b_1 and b_2. Since we assumed $\mathcal{A}$ to be convex, $O_{\mathcal{A}}$ lies within the convex angular sector bounded by the two half-lines supporting E_1 and E_2 and erected from a. The orientation ϕ_1 (resp. ϕ_2) is achieved when the edge E_1 (resp. E_2) is aligned with the edge F.
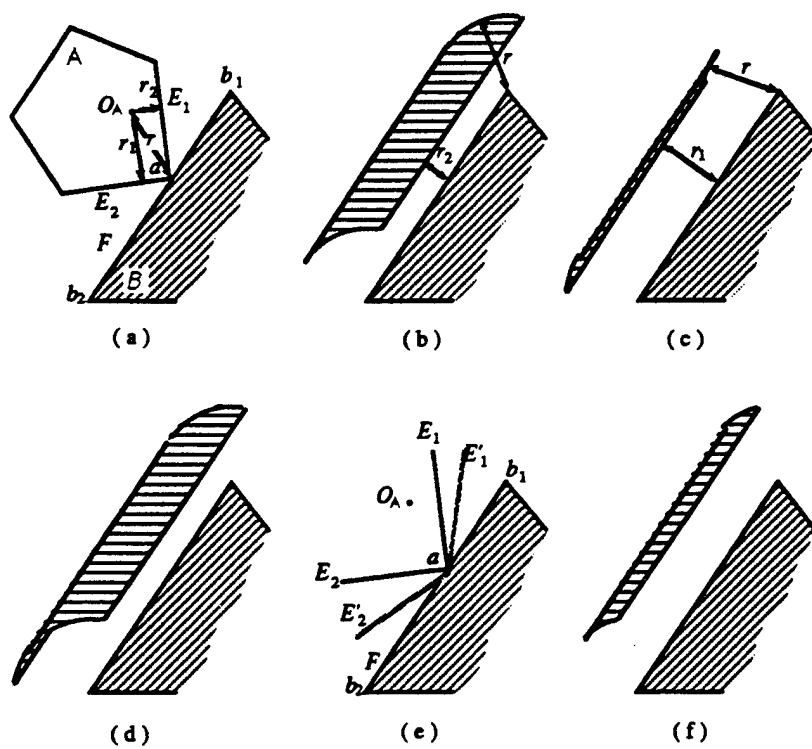
Figure 2.8: Projection of a Type B C-Facet

Assuming that $[\phi_1, \phi_2] \subseteq [\gamma_u, \gamma_{u+1}]$, the projection of e_B on the xy plane is shown in Figure 2.8d. It is obtained as the union of two regions, shown in Figures 2.8b and 2.8c.

Let ψ be the orientation of $\mathcal{A}$ when a lies in F and the segment $\overline{aO_\mathcal{A}}$ is perpendicular to F. The region in Figure 2.8b is the locus of $O_\mathcal{A}$, when $\mathcal{A}$ translates and rotates with the vertex a staying in F and the orientation θ ranging over $[\phi_1, \psi]$. (If $\psi < \phi_1$, then the region is empty.) This region is bounded by two circular arcs and two straight segments. The two arcs are centered at b_1 and b_2, respectively, and have the same radius equal to the distance between a and $O_\mathcal{A}$. The region in Figure 2.8c is the locus of $O_\mathcal{A}$, when $\mathcal{A}$ translates and rotates with a staying in F and the orientation θ ranging over $[\psi, \phi_2]$. (If $\psi > \phi_2$, the region is empty.)

The case where $[\gamma_u, \gamma_{u+1}] \subset [\phi_1, \phi_2]$ can be treated in the same way, by drawing fictitious edges E_1' and E_2' from vertex a (see Figures 2.8e and 2.8f), so that, when E_1' (resp. E_2') is aligned with F, $\mathcal{A}$'s orientation is γ_u (resp. γ_{u+1}).

Computation of $\mathcal{OCB}_{xy}[\kappa^u]$ and $\mathcal{ICB}_{xy}[\kappa^u]$. The projection of every C-facet is a generalized polygon with a small number of edges – two straight segments and two circular arcs. The union of all the generalized polygons forms a "donut" shaped region (see Figure 2.6).

Γ_1 and Γ_2 can be extracted by a line-sweep technique. This is a well-known technique in Computational Geometry (e.g., see [Preparata and Shamos, 1985]). It consists of sweeping a line across the plane. For example, the line is parallel to the x axis and is swept bottom-up. At each instant, the "status" of the line – i.e., the list of the intersections of the line with the generalized polygons – is represented in a balanced tree [Aho, Hopcroft and Ullman, 1983]. The status of the line changes in a qualitative fashion only at a finite number of ordinates, called "events", where the line is either tangent to a C-obstacle or passing through the intersection of two generalized polygons. At every event, the list of future events, which is also represented in a balanced tree, is updated.

The line-sweep process starts at the bottom-most ordinates of all the points in the

generalized polygons. Below this ordinate, the line intersects no generalized polygon and thus lies entirely outside $\mathcal{OCB}_{xy}[\kappa^u]$. During the sweeping process, the contours Γ_1 and Γ_2 are tracked by labelling the intervals between the ordinates listed in the sweep-line status as being outside $\mathcal{OCB}_{xy}[\kappa^u]$, inside $\mathcal{OCB}_{xy}[\kappa^u]$ but outside $\mathcal{ICB}_{xy}[\kappa^u]$, or inside $\mathcal{ICB}_{xy}[\kappa^u]$.

Once Γ_1 and Γ_2 have been extracted in the form of sequences of straight segments and circular arcs, it is not difficult to clip them by the rectangle $[x_1, x_2] \times [y_1, y_2]$. The projections $\mathcal{OCB}_y[\kappa^{uv}]$ and $\mathcal{ICB}_y[\kappa^{uv}]$ are easily computed by determining the points of Γ_1 and Γ_2 at abscissae a_v, $v = 1, ..., s+1$, and the other extremal points of Γ_1 and Γ_2 within each interval $[a_v, a_{v+1}]$, $v = 1, ..., s$. In fact, all these computations can be done during line sweeping. If the interval $[x_1, x_2]$ is decomposed into sub-intervals, the sweep line has to be parallel to the x axis, otherwise it should be parallel to the y axis.

The overall line-sweep process takes time $O((n+m)\log n)$, where n is the number of C-facets that intersect with the interval $[\gamma_u, \gamma_{u+1}]$ and m is the number of intersections of the generalized polygons. We know that $n \leq n_A n_B$ (see Subsection 2.1.1). On the other hand, $m \in O(n^2)$, but it is usually much smaller.

One way to improve the efficiency of the algorithm could be to restrict the line-sweep process to the rectangle $[x_1, x_2] \times [y_1, y_2]$. However, there seems to be no simple way of establishing the initial status of the sweeping line if it does not start at the bottom-most ordinate (or left-most abscissa). Nevertheless, the process can be stopped as soon as the sweeping line leaves the rectangle.

Generalization. If $\mathcal{A}$ is a non-convex polygon that can be decomposed into convex components, such that the interiors of all these components have a non-empty intersection, then the reference point can be selected within this intersection and the above method applies directly to each component taken separately.

If $\mathcal{A}$ is non-convex and cannot be represented as the union of overlapping convex components (for instance, it is a ⊔-shaped object), the above method can still be applied, but with some changes. Since the reference point will be outside some of the

convex components, the shape of the projection of the corresponding C-facets will be different, but not difficult to establish.

Avnaim and Boissonnat [Avnaim and Boissonnat, 1988] describe an algorithm of time complexity $O(n_A^3 n_B^3 \log n_A n_B)$ for computing the description of the boundary of CB, when both A and B are non-convex. In the case where A is non-convex, using this algorithm first and next projecting the C-obstacles on the xy plane, would probably be an efficient method. However, we have not implemented it.

2.2.6 Swept-Area Method

Principle. The swept-area method consists of first computing two areas swept out by A when it rotates about its reference point from orientation γ_u to orientation γ_{u+1}: the *outer swept area* denoted by $OSA[\gamma_u, \gamma_{u+1}]$ and the *inner swept area* denoted by $ISA[\gamma_u, \gamma_{u+1}]$. They are defined as follows:

$$OSA[\gamma_u, \gamma_{u+1}] = \bigcup_{\theta \in [\gamma_u, \gamma_{u+1}]} A(0,0,\theta) = \{(x,y) \mid \exists \theta \in [\gamma_u, \gamma_{u+1}] : (x,y) \in A(0,0,\theta)\},$$

$$ISA[\gamma_u, \gamma_{u+1}] = \bigcap_{\theta \in [\gamma_u, \gamma_{u+1}]} A(0,0,\theta) = \{(x,y) \mid \forall \theta \in [\gamma_u, \gamma_{u+1}] : (x,y) \in A(0,0,\theta)\}.$$

Then, the method regards both $OSA[\gamma_u, \gamma_{u+1}]$ and $ISA[\gamma_u, \gamma_{u+1}]$ as moving objects, which can only translate in the plane, and it maps the obstacle B in the configuration spaces, $\mathbf{R}^2$, of these objects. The mapping is obtained by "growing" B inversely to the shape of these two objects, leading to two regions $OB[\gamma_u, \gamma_{u+1}]$ and $IB[\gamma_u, \gamma_{u+1}]$ formally defined as follows (see Subsection 2.1.1):

$$OB[\gamma_u, \gamma_{u+1}] = B \ominus OSA[\gamma_u, \gamma_{u+1}]$$
$$IB[\gamma_u, \gamma_{u+1}] = B \ominus ISA[\gamma_u, \gamma_{u+1}].$$

It is shown in [Laumond, 1987] that the Minkowski sum (resp. difference) of two generalized polygons is a generalized polygon, which can be computed in time $O(n_1 + n_2)$, if the two input polygons are convex, and $O(n_1 n_2)$, otherwise. (n_1 and n_2 denote the number of edges of the input generalized polygons.)

The swept-area method returns two regions:

$$[x_1, x_2] \times [y_1, y_2] \cap OB[\gamma_u, \gamma_{u+1}]$$

and:

$$[x_1, x_2] \times [y_1, y_2] \cap \mathcal{IB}[\gamma_u, \gamma_{u+1}].$$

The first is exactly $\mathcal{OCB}_{xy}[\kappa^u]$, as shown in [Lozano-Pérez, 1983]. The second is strictly included in $\mathcal{ICB}_{xy}[\kappa^u]$, which leads the overall decomposition algorithm to generate a set of FULL cells of less total volume than with the projection method.

In practice, the swept-area method is significantly more efficient than the projection method when the intervals $[\gamma_u, \gamma_{u+1}]$ are large. When these intervals become small enough, the projection method is preferred because it exactly computes $\mathcal{ICB}_{xy}[\kappa^u]$.
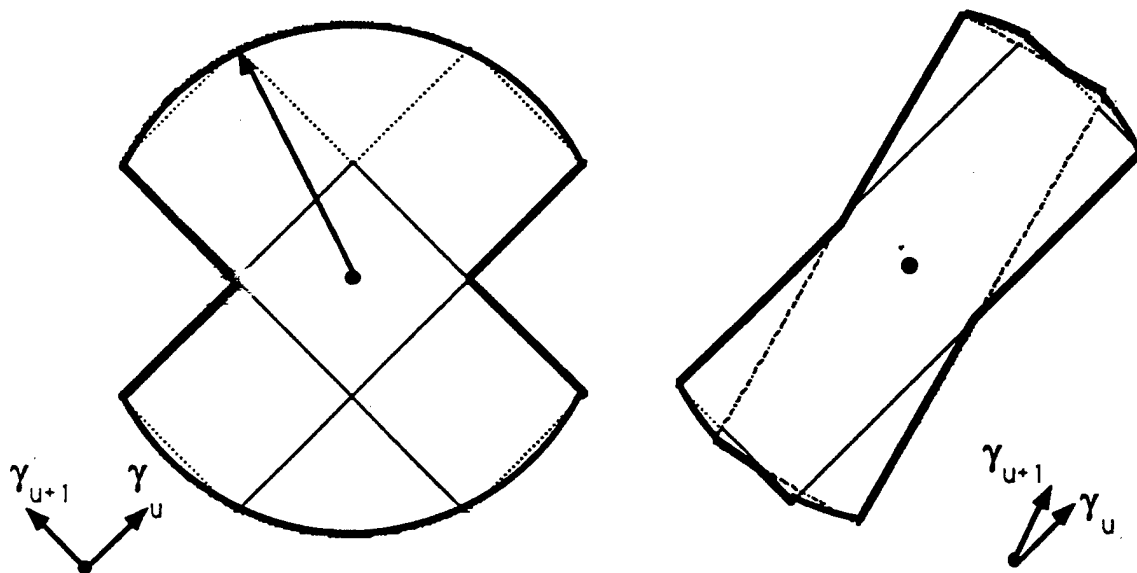
Figure 2.9: The contour of the outer swept area of $\mathcal{A}$ when it rotates about its reference point from orientation γ_u to orientation γ_{u+1}

Computation of $\mathcal{OSA}[\gamma_u, \gamma_{u+1}]$. The outer swept area $\mathcal{OSA}[\gamma_u, \gamma_{u+1}]$ is a generalized polygon (see Figure 2.9). The straight edges of this polygon are portions of edges of $\mathcal{A}(0, 0, \gamma_u)$ and $\mathcal{A}(0, 0, \gamma_{u+1})$. Each circular edge is the locus of a vertex of $\mathcal{A}$, when $\mathcal{A}$ rotates from γ_u to γ_{u+1} about the reference point.

The contour of $\mathcal{OSA}[\gamma_u, \gamma_{u+1}]$ is traced out in time $O(n_\mathcal{A}^2)$ starting at the vertex that is farthest from $O_\mathcal{A}$.
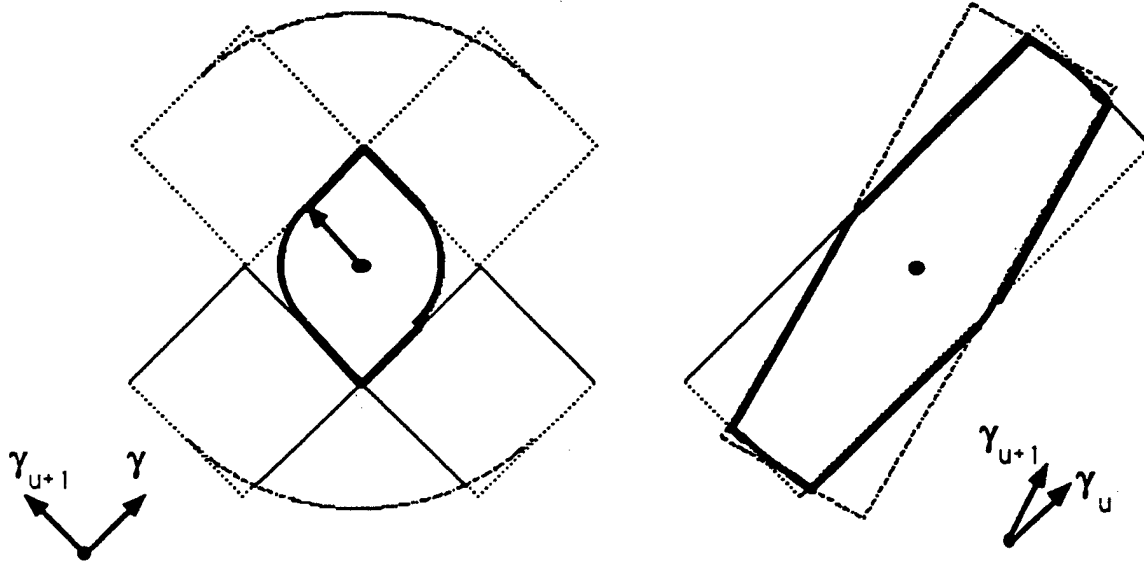
Figure 2.10: The contour of the inner swept area of $\mathcal{A}$ when it rotates about its reference point from orientation γ_u to orientation γ_{u+1}

Computation of $\mathcal{ISA}[\gamma_u, \gamma_{u+1}]$. The inner swept area $\mathcal{ISA}[\gamma_u, \gamma_{u+1}]$ is also a generalized polygon (see Figure 2.10). The straight edges of this polygon are portions of the edges of $\mathcal{A}(0, 0, \gamma_u)$ and $\mathcal{A}(0, 0, \gamma_{u+1})$. Each circular edge is the locus of a point obtained by projecting the reference point on an edge of $\mathcal{A}$, if that projection falls between the two extremities of the edge.

The contour of $\mathcal{ISA}[\gamma_u, \gamma_{u+1}]$ can be traced out in time $O(n_\mathcal{A}^2)$. The starting point is the first intersection of a ray (drawn from the reference point) with the potential edges of $\mathcal{ISA}[\gamma_u, \gamma_{u+1}]$. (Since the reference point is in the interior of $\mathcal{A}$, it is also in the interior of $\mathcal{ISA}[\gamma_u, \gamma_{u+1}]$.)

2.3 Construction of a Channel

2.3.1 First-Cut Algorithm

Remember from Subsection 2.1.3 that a channel is a sequence of adjacent EMPTY or MIXED cells connecting the initial configuration $\mathbf{q}_{init} = (x_{init}, y_{init}, \theta_{init})$ to the goal configuration $\mathbf{q}_{goal} = (x_{goal}, y_{goal}, \theta_{goal})$. A channel is EMPTY, if it contains only EMPTY cells; otherwise, it is MIXED. Below, we call an EMPTY channel an **E-channel**, and a MIXED channel a **M-channel**.

A simple first-cut search algorithm for generating an E-channel is the following:

1. Generate a first partition $\mathcal{P}_0$ of $\mathcal{K}$. Construct the graph CG_0 corresponding to this decomposition. Set i to 0.

2. Search CG_i for a channel. If an E-channel is found, return success. If no channel is found, return failure.

3. Let Π be the M-channel generated at Step 2. Set $\mathcal{P}_{i+1}$ to $\mathcal{P}_i$ and i to $i + 1$. For every MIXED cell κ in Π, partition κ into a set $\mathcal{P}_\kappa$ of smaller cells and set $\mathcal{P}_i$ to $[\mathcal{P}_i \backslash \kappa] \cup \mathcal{P}_\kappa$. Goto Step 2.

This algorithm searches successive cg's until an E-channel is found. Each cg CG_i, $i \neq 0$, is obtained from the previous cg, i.e. CG_{i-1}, by expanding only the MIXED nodes that belong to the M-channel generated in CG_{i-1}.

The search for a channel in a cg may be guided by various types of heuristics. In general, a cg is not necessarily searched for an E-channel before it is searched for a M-channel. Indeed, although it is natural that the heuristics put an extra cost on MIXED cells in order to generate an E-channel quicker, it may also be appropriate to prefer shorter channels over longer ones (according to some metrics). Thus, although an E-channel may exist in a cg CG_i, it may be preferable to generate a significantly shorter M-channel instead, and refine CG_i accordingly. Notice that any E-channel existing in CG_i still exists in all its successors CG_{i+j}, $j = 1, 2, \ldots$.

2.3.2 Improved Algorithm

The major drawback of the simple first-cut algorithm given above is that the search work performed in CG_i, if it does not return success, is not used to help the search of CG_{i+1}. This drawback can be remedied as follows. Rather than reconstructing a full cg, whenever MIXED cells along a M-channel are refined, a cg representing the decomposition of every refined cell κ is generated separately and recursively searched for a "subchannel". This subchannel is a sequence of adjacent EMPTY or MIXED cells produced by the decomposition of κ.

The new algorithm hence generates a hierarchy of cg's. The cg at the top of the hierarchy corresponds to the initial decomposition of C – i.e., K – and is denoted by CG_C. Every other cg corresponds to the decomposition of a certain MIXED cell, say κ, and is denoted by CG_κ. A channel Π is first generated in CG_C. If Π is an E-channel, the planner exits with success; otherwise, each MIXED cell κ in Π is decomposed recursively, and a subchannel Π_κ, if any, is generated in CG_κ. This subchannel is substituted for κ in Π.

In order to make the algorithm work properly, however, one must be careful that each subchannel Π_κ connects appropriately to the rest of Π [Kambhampati and Davis, 1986]. This can be worked out as explained below, by generating a complete channel connecting q_{init} to q_{goal} at every level of refinement.

Let Π^1 be an M-channel extracted from the top-level cg, i.e. CG_C. Set Π^2 to the empty sequence of cell. (Π^2 will be incrementally augmented into a refinement of Π^1.) Each cell κ in Π^1 is considered in the order it appears in Π^1. If κ is EMPTY, it is simply appended to the current Π^2. If κ is MIXED, it is first decomposed into a set $\mathcal{P}_\kappa$ of smaller cells and a cg CG_κ is built using this decomposition; then, CG_κ is searched for a subchannel Π_κ satisfying the following four conditions:

- If κ is the first cell in Π^1 (hence, it contains q_{init}), then the first cell in Π_κ also contains q_{init}.

- If κ is the last cell in Π^1 (hence, it contains q_{goal}), then the last cell in Π_κ also contains q_{goal}.

-· If κ is not the first cell in Π^1, the first cell of Π_κ is adjacent to the last cell of the current Π^2.

- If κ is not the last cell in Π^1, the last cell of Π_κ is adjacent to the cell following κ in Π^1.

In the following, a sequence of adjacent EMPTY or MIXED cells in the decomposition of κ is called a **subchannel** iff it satisfies these conditions.

Assuming that the planner succeeds in generating Π_κ, Π^2 is modified by appending Π_κ to it. The case when the planner fails in generating Π_κ will be examined in Subsection 2.3.4.

When all the cells in Π^1 have been considered successfully, we have obtained a channel Π^2, which is a refinement of Π^1. If Π^2 is an E-channel, the planning problem is solved; the planner returns Π^2 and success. Otherwise, Π^2 is recursively refined into a third channel Π^3 by decomposing the MIXED cells it contains. Etc...

2.3.3 Cell Occurrences in a Channel

An E-channel is now generated by refining a M-channel without reproducing a global cg. Therefore, it is important that we allow a M-channel to contain the same MIXED cell several times (i.e., loops). Indeed, different occurrences of the same MIXED cell may lead to different subchannels. This need is better explained using an example.

Consider the two-dimensional example of Figure 2.11. In Figure 2.11a, the sequence of cells $(\kappa_1, \kappa_2, \kappa_3, \kappa_2, \kappa_4)$, which contains κ_2 twice, has to be considered as a M-channel. (The grey regions are C-obstacles, hence κ_2 and κ_3 are MIXED, while κ_1 and κ_4 are EMPTY.) We assume that κ_2 and κ_3 are decomposed as shown in Figure 2.11b. When the two occurrences of κ_2 are refined, they produce different EMPTY subchannels. This is illustrated in Figure 2.11b, where κ_2 is partitioned into three smaller cells with two EMPTY cells (κ_{21} and κ_{22}) and one FULL cell (κ_{23}). The first occurrence of κ_2 leads to a one-cell subchannel made of κ_{21}, while the second occurrence leads to another one-cell subchannel made of κ_{22}. On the other hand, the
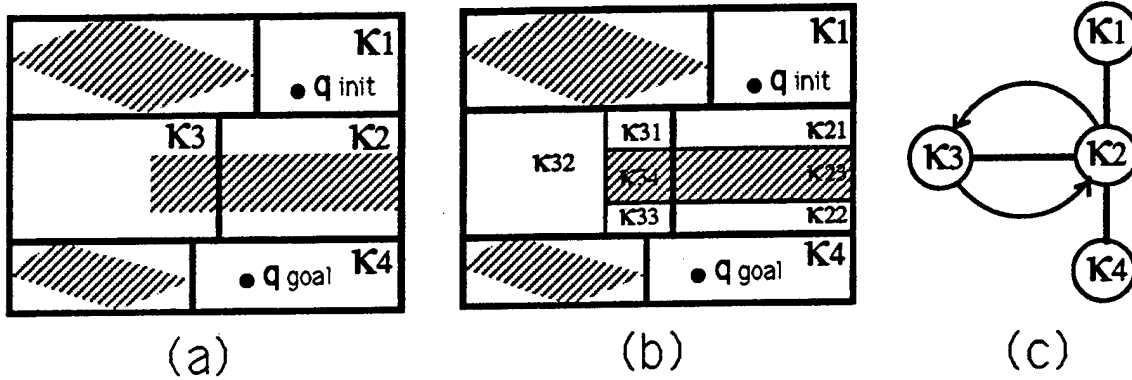
Figure 2.11: Multiple Cell Occurrences in a M-channel

decomposition of κ_3 produces four cells with three EMPTY cells (κ_{31}, κ_{32}, and κ_{33}) and one FULL cell (κ_{34}). From this decomposition the EMPTY subchannel ($\kappa_{31}, \kappa_{32}, \kappa_{33}$) is extracted. The generated E-channel is ($\kappa_1, \kappa_{21}, \kappa_{31}, \kappa_{32}, \kappa_{33}, \kappa_{22}, \kappa_4$).

Since the same cell may appear several times in a channel Π^k, we now refer to the elements in a channel as **cell occurrences** rather than just cells. We denote by ω_i^k the ith cell occurrence in Π^k and $cell(\omega)$ the cell of which ω is an occurrence. A cell occurrence ω is said to be MIXED (resp. EMPTY) iff the cell $cell(\omega)$ is MIXED (resp. EMPTY). A subchannel generated as a refinement of a MIXED cell occurrence ω is denoted by Π_ω; it is constructed by searching $CG_{cell(\omega)}$.

In Subsection 2.3.6, we examine the impact of allowing loops in a channel on the search of a cg.

2.3.4 Failure Recovery

Consider now the situation where the planner is constructing the channel Π^{i+1} and fails to refine a MIXED cell occurrence ω_i^j contained in Π^i into a subchannel. We have mentioned this situation in Subsection 2.3.2. It may be caused either by the fact that Π^i does not contain an E-channel as illustrated in Figure 2.12, or by the fact that the partial channel Π^{i+1} generated so far is a blind alley as illustrated in Figure 2.13.

Figure 2.12: First Type of Failure

Figure 2.12a shows a M-channel $\Pi^1 = (\kappa_1, \kappa_2, \kappa_3)$ in bold lines. Assume that κ_1 and κ_2 are decomposed as shown in Figure 2.12b. The subchannel $(\kappa_{11}, \kappa_{12})$ is constructed within κ_1. But then the planner fails to find a subchannel within κ_2 because a C-obstacle obstructs the passage between κ_1 and κ_3 through κ_2. Recovering from this failure requires the planner to generate an alternative channel $\Pi^{1\prime}$.

Figure 2.13a shows a M-channel $\Pi^1 = (\kappa_1, \kappa_2, \kappa_3, \kappa_8)$. Assume that κ_1, κ_2 and κ_3 are decomposed as shown in Figure 2.13b, and that the incomplete channel $\Pi^2 = (\kappa_{11}, \kappa_{12}, \kappa_{21}, \kappa_{22}, \kappa_{23})$ has been generated. But, the planner fails to find a subchannel in the decomposition of κ_3. Recovering from this failure requires the planner to generate alternative subchannels in κ_1 and κ_2.

Figure 2.13: Second type of failure

Of course, the planner has no way to directly determine the cause of the failure. Instead, when it fails to refine the MIXED cell occurrence ω_i^j into a subchannel, it distinguishes among the three cases described below which are mutually exclusive:

Case (1): $j = 1$.

$\omega_i^j = \omega_i^1$ is the first cell occurrence in Π^i. The failure typically means that there is no way to connect q_{init} to the second cell occurrence in Π^i. The only alternative for the planner is to attempt to generate a new channel Π^i. If no new channel Π^i can be generated and $i > 1$, the planner recursively attempts to generate a new channel Π^{i-1}. If no new channel Π^i can be generated and $i = 1$, the planner returns failure.

Case (2): $j > 1$ and ω_i^{j-1} is EMPTY.

The failure typically means that there is no way to connect any configuration in the EMPTY cell occurrence ω_i^{j-1} to a configuration in ω_i^{j+1} through ω_i^j. The planner then proceeds as described in case (1). (Some differences between these two cases will be made apparent in the next subsection.)

Case (3): $j > 1$ and ω_i^{j-1} is MIXED.

The failure typically means that there is no way to connect any configuration in the last cell occurrence of the current Π^{i+1} (which is a subset of ω_i^{j-1}) to a configuration in ω_i^{j+1} through ω_i^j. The planner attempts to generate another subchannel $\Pi_{\omega_i^{j-1}}$ in the cg of $cell(\omega_i^{j-1})$. If it finds one, the planner substitutes it for the previous one in Π^{i+1}, and tries again to refine ω_i^j into a subchannel by searching the cg of $cell(\omega_i^j)$. Otherwise, it iteratively treats ω_i^{j-1} just as it treated ω_i^j, i.e. it sets j to $j - 1$ and applies the treatment of case (1), (2), or (3), whichever is applicable.

2.3.5 Recording Failure Conditions

One can organize (sub)channel generation and failure recovery in a systematic fashion in order to avoid infinite looping. However, this may not prevent the planner from making the same mistakes several times. Higher efficiency can be obtained by remembering the conditions of the failures in order to avoid reproducing them [Stallman and Sussman, 1977] [Latombe, 1979]. One way to remember failure conditions is to annotate cells as follows, where these annotations are used to guide the future search of channels. Cases (1), (2), and (3) below refer to the same cases as in the previous subsection. n_k denotes the number of cell occurrences in Π^k.

Case (1):

Let $\xi = cell(\omega_1^k)$ and $\psi = cell(\omega_2^k)$ in the current Π^k. If $n_k = 1$, by convention $\psi = \lambda$, where λ denotes the nonexisting cell. The cell ξ is annotated with $A1 : [\psi]$, which is called a *type 1 annotation*. This annotation is constructed when a case 1 failure has occurred in ω_1^k without having a case 3 failure occurring in ω_2^k.

This annotation is later used as follows: If a new channel Π^k is generated, which contains an occurrence ω_ξ of ξ in the first position, then an occurrence of ψ should not be considered a valid successor of ω_ξ. In the particular case where $\psi = \lambda$, this means that ω_ξ must not be the last cell occurrence of the new Π^k.

Notice that the first cell occurrence in the new Π^k is necessarily an occurrence of ξ, since ξ contains q_{init}. But there may be other occurrences of ξ appearing in this new Π^k, to which the above annotation does not apply.

Case (2):

Let $\xi = cell(\omega_i^k)$ $(i > 1)$, $\varphi = cell(\omega_{i-1}^k)$, and $\psi = cell(\omega_{i+1}^k)$ in the current Π^k. If $i = n_k$, then $\psi = \lambda$ (with the same meaning of λ as above). ξ is annotated with $A2 : [\varphi, \psi]$, which is called a *type 2 annotation*. This annotation is constructed when a case 2 failure has occurred in ω_i^k $(i > 1)$ without having a case 3 failure occurring in ω_{i+1}^k.

This annotation is later used as follows: If a new channel Π^k is generated, which contains an occurrence ω_ξ of ξ preceded by an occurrence of φ, then an occurrence of ψ should not be considered a valid successor of ω_ξ. In the particular case where $\psi = \lambda$, this means that ω_ξ must not be the last cell occurrence of the new Π^k.

Notice that if $\psi \neq \lambda$, the construction is symmetrical, i.e., if ξ cannot be traversed by a subchannel connecting φ to ψ, neither can it be traversed by a subchannel connecting ψ to φ. Therefore, the annotation $A2 : [\varphi, \psi]$ is later used commutatively. This annotation can be extended to record the failure condition where a case 2 failure occurs in ω_i^k after one or more case 3 failures has occurred in ω_{i+1}^k.

Case (3):

In this case, failure conditions are more involved than in cases (1) and (2), and may require sequences of cells of arbitrary length to be considered. The cell $\xi = cell(\omega_{i-1}^k)$ is annotated with an expression of the form $A3 : [\varphi, (\psi_1\psi_2...\psi_q)]$, called a *type 3 annotation*. This annotation is constructed when a case 3 failure has occurred in ω_i^k. The detailed construction will be explained below.

This annotation is later used as follows: if a subchannel Π_{ω_ξ} is generated in an occurrence ω_ξ of ξ, while ω_ξ is followed in the current channel Π^k by a sequence $\omega_1, \omega_2, ..., \omega_q$ of cell occurrences, such that for all $j \in [1, q]$ $cell(\omega_j) = \psi_j$ (if $\psi_q = \lambda$, the condition $cell(\omega_q) = \lambda$ requires that ω_{q-1} be the last cell occurrence in the current Π^k), then the last cell occurrence in Π_{ω_ξ} must not be an occurrence of φ.

Let us now examine how the annotation was built. Let $\xi = cell(\omega_{i-1}^k)$, $\psi_1 = cell(\omega_i^k)$, and a case 3 failure occur in ω_i^k. Let φ be the last cell occurrence in the (incomplete) current Π^{k+1}, i.e. φ is the last cell occurrence of the subchannel $\Pi_{\omega_{i-1}^k}$ previously generated. For every type 3 annotation $A3 : [\varphi', (\psi_1'...\psi_p')]$ attached to ψ_1, if ω_i^k is followed in the current Π^k by the sequence $\omega_{i+1}^k, ..., \omega_{i+p}^k$, with $cell(\omega_{i+j}^k) = \psi_j'$ for all $j \in [1, p]$, then ξ is annotated with $A3 : [\varphi, (\psi_1\psi_1'...\psi_p')]$. If no type 3 annotation is currently attached with ψ_1, then ξ is annotated with $A3 : [\varphi, (\psi_1\psi_2)]$ where $\psi_2 = cell(\omega_{i+1}^k)$ if $i < n_k$ or $\psi_2 = \lambda$ if $i = n_k$.

2.3.6   Search of a Cell-Connectivity Graph

The planning algorithm builds a hierarchy of cg's and searches each of these cg's separately. The cg at the top of the hierarchy is CG_C and is searched to generate the channel Π^1. Every other cg CG_κ represents the decomposition $\mathcal{P}_\kappa$ of a cell κ which belongs to the parent cg of CG_κ in the cg hierarchy.

Remember that $\mathcal{C}$ is just a particular cell $\mathcal{K}$. We denote by Π^0 the channel that only contains ω_1^0, with $cell(\omega_1^0) = \mathcal{K}$. Π^1 is a refinement of Π^0. In the very particular case where there is no C-obstacle in the configuration space, then $\mathcal{K}$ is an EMPTY cell, and Π^0 is an E-channel, so that Π^1 does not have to be generated.

Let us consider that we have generated a M-channel Π^k ($k \geq 0$), which we now

refine into Π^{k+1}, and that we are currently considering the MIXED cell occurrence $\omega_i^k \in \Pi^k$. Let $\kappa = cell(\omega_i^k)$. If κ has already been decomposed, we just reuse the cg CG_κ that was previously generated; otherwise, we decompose κ into a set $\mathcal{P}_\kappa$ of smaller cells using the method described in Section 2.2 and we build CG_κ. We then search CG_κ for a subchannel that will be appended to the tail of the current Π^{k+1}.

The search of CG_κ consists of first determining the possible initial and goal cells of the subchannel to be generated:

- If $i = 1$, ω_i^k is the first cell of Π^k. Then the only possible initial cell in CG_κ is the cell of $\mathcal{P}_\kappa$ that contains $\mathbf{q}_{init}$. Otherwise, the possible initial cells of CG_κ are all the cells of $\mathcal{P}_\kappa$ which are adjacent to the cell – call it φ – of the last cell occurrence ω_φ in the current Π^{k+1}, and whose insertion in Π^{k+1} as cell occurrences succeeding ω_φ does not violate any annotation of type 1 or 2 attached to φ.

- If $i = n_k$, ω_i^k is the last cell occurrence of Π^k. The only possible goal cell in CG_κ is the cell of $\mathcal{P}_\kappa$ that contains $\mathbf{q}_{goal}$. Otherwise, the possible goal cells of CG_κ are all the cells of $\mathcal{P}_\kappa$ which are adjacent to $cell(\omega_{i+1}^k)$, and which do not violate any annotation of type 3 attached to κ.

If there is no possible initial or goal cell in CG_κ, the planner considers the situation as a failure to refine ω_i^k and applies the treatment described in Subsections 2.3.4 and 2.3.5.

If at least one initial and one goal cell are established in CG_κ, the planner searches the graph for a subchannel $\Pi_{\omega_i^k}$ linking any of the initial cells to any of the goal cells. It does this in a way such that no annotation of type 1 or 2 attached to the cells in the generated subchannel is violated.

As noted previously, the subchannel $\Pi_{\omega_i^k}$ to be generated in CG_κ should be allowed to contain several occurrences of any MIXED cell in $\mathcal{P}_\kappa$. However, it seems quite realistic in an implementation to limit the number of occurrences of the same cell in a subchannel[6].

---

[6]In our implementation, this number is not bounded. However, we apply a best-first search strategy, which always considers the (sub)channels of minimal "cost" first. The cost of a (sub)channel grows with its length and the number of MIXED cells it contains.

Since there may be multiple occurrences of the same cell in a subchannel, the actual graph that is searched to refine a cell occurrence ω_i^k is not CG_κ, but a graph derived from it. This graph contains the subset of subchannels (with and without loops) in CG_κ, which do not include two occurrences of the same EMPTY cell or more occurrences of the same MIXED cell than a predefined number (see paragraph above). This search graph is made (partly) explicit while it is searched. Since a MIXED cell is adjacent to a finite number of other cells, the search graph is finite. Hence its search is guaranteed to terminate, either with success or with failure. If it terminates with success, the generated subchannel $\Pi_{\omega_i^k}$ is appended to Π^{k+1}. If it terminates with failure, the treatment described in Subsections 2.3.4 and 2.3.5 is applied. Thanks to the annotations attached to the cells, the search cannot iterate forever through the same sequence of failures.

2.3.7 Boundary-Connectivity Graph

Cell-connectivity graphs are one way to represent connectivity among cells. Another way is to build a graph called **boundary-connectivity graph**, or **bcg**. Given a partition $\mathcal{P}_\kappa$ of a cell κ into non-overlapping cells, the bcg BCG_κ is defined as follows:

- Each node of BCG_κ corresponds to the intersection of the boundaries of two adjacent cells.

- Two nodes of BCG_κ are connected by a link iff the two nodes are portions of the boundary of the same cell.

Each link in a bcg can be regarded as the connection between two cells (each cell includes the boundary portion associated with one extremity of the link) through a third one.

For the same cell κ, the size of BCG_κ is larger than that of CG_κ. However, bcg's present some advantages over cg's because they represent more explicitely the connectivity of the cells in $\mathcal{P}_\kappa$. In particular, using bcg's removes the need for type 1 and 2 annotations. Indeed, attaching a type 1 or 2 annotation to a cell in a cg is equivalent to removing a link in a bcg. However, in order to use bcg's properly one

must be careful about various details. In particular, the initial and goal configurations should appear as nodes in some of the bcg's.

One possible way to use bcg's is sketched below. Every M-channel Π^{k-1} ($k > 0$) is still a sequence of cell occurrences. When it is refined into Π^k, an *extended bcg BCG^k* is constructed as the bcg of all the cells which are either EMPTY cells in Π^k or cells in the decomposition $\mathcal{P}_\kappa$ of a MIXED cell κ in Π^k. In addition, $\mathbf{q}_{init}$ and $\mathbf{q}_{goal}$ are included in BCG^k as nodes. The node corresponding to $\mathbf{q}_{init}$ (resp. $\mathbf{q}_{goal}$) is connected by a link to every node representing a boundary portion of the cell containing $\mathbf{q}_{init}$ (resp. $\mathbf{q}_{goal}$). When Π^k is refined, parts of BCG^k may already exist and are simply re-used. The links having $\mathbf{q}_{init}$ as one extremity are called *initial links*. The links having $\mathbf{q}_{goal}$ as one extremity are called *goal links*. The links contained in the bcg BCG_κ of a cell κ are called *intra-links*. All the other links are called *inter-links*.

The refinement of Π^{k-1} into Π^k is essentially done in the same way as before. The only difference is that, rather than searching the cg of the MIXED cell occurrences along Π^{k-1}, the planner now searches the portion of BCG^k that corresponds to the bcg of the MIXED cell occurrence currently being considered. The initial links, goal links, and inter-links are used only to determine the possible initial and goal nodes of this search. Case (1) and (2) failures lead to the removal of some links from BCG^k.

2.4 Implementation and Experimentation

We have implemented the techniques described in the above sections in a path planner. The planner is written in Allegro Common Lisp and runs on an Apple Macintosh II computer.

We ran the planner on many examples including both convex and non-convex moving objects. Twelve of these examples are illustrated in Figures 2.14 and 2.15. In these figures, we show a path extracted from the generated channel by connecting the center points of the entrance and exit of each cell along the channel by a straight line segment. Statistics – CPU time, total number of cells generated (N_{tot}), number of
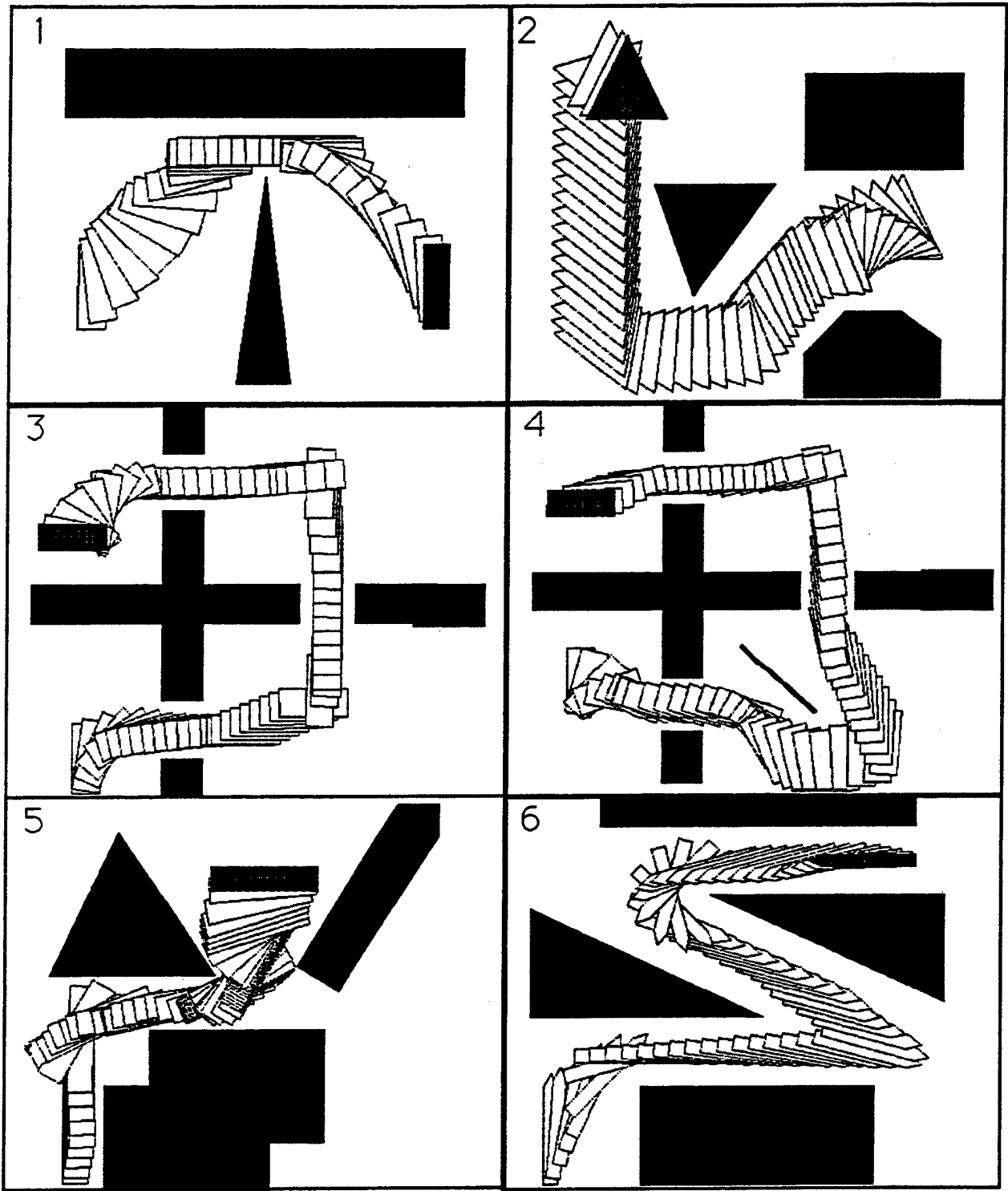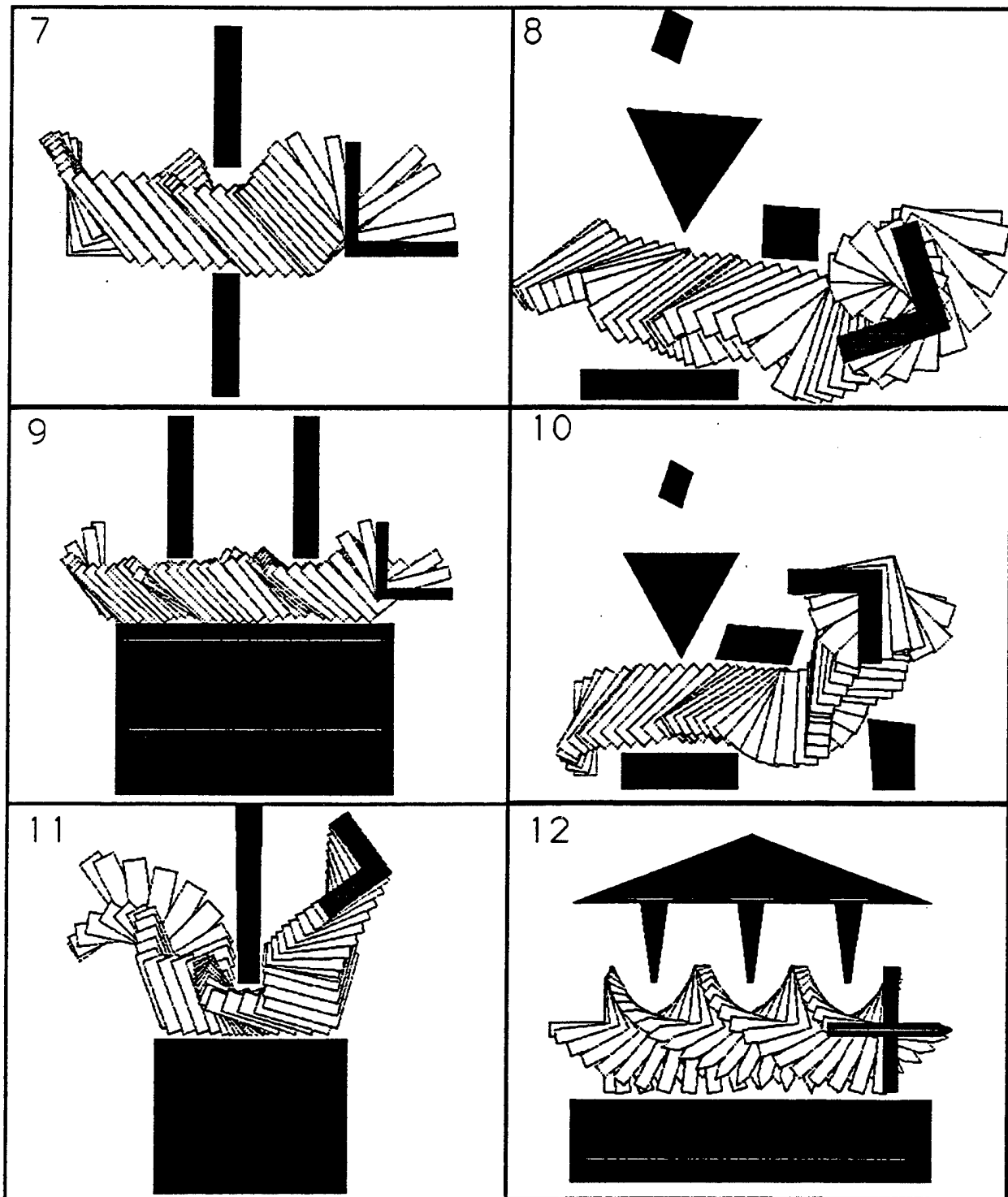
Figure 2.14: Motion planning examples 1 - 6

Figure 2.15: Motion planning examples 7 - 12

EMPTY cells generated (N_{EMPTY}), number of EMPTY cells used in the output channel (N_{used}), and decomposition efficiency measure (E) – characterizing the efficiency of the planner are given in Table 1.

example	CPU time (min)	N_{total}	N_{EMPTY}	N_{used}	E ($\times 1000$))
1	0.6	98	35	12	9.4
2	0.9	140	74	18	8.0
3	1.5	210	104	10	6.9
4	2.5	264	134	28	5.9
5	5.5	293	96	36	4.8
6	5.0	218	116	29	5.3
7	0.9	160	77	22	12.0
8	2.5	205	88	17	6.1
9	6.5	170	25	13	9.8
10	9.8	206	46	19	2.1
11	11.0	312	72	21	7.9
12	10.5	369	121	18	4.3

Table 2.1: Statistics for the twelve examples

	Example 5				Example 8			
planner	CPU time	N_{total}	N_{EMPTY}	N_{used}	CPU time	N_{total}	N_{EMPTY}	N_{used}
Ours	5.5	293	96	36	2.5	205	88	17
BLP	tens	644	120	87	tens	782	62	29

Table 2.2: Comparision with the BLP planner

Examples 5 and 8 have been previously reported in [Brooks and Lozano-Pérez, 1982] (BLP planner), the best previous planner known to the authors using the hierarchical approximate cell decomposition approach. In Table 2, we give the comparison of the statistics on these two examples with our planner and BLP planner[7]. We should point out that for Example 5 the path generated by our planner is different from the path generated by BLP planner, i.e., it does not have a backing-up maneuver.

---

[7]The BLP planner was implemented on a LISP machine, which runs LISP approximately 5 times faster than the Macintosh II runs Allegro Common Lisp (according to our own benchmark). In addition, since we do not count the number of FULL cells in our planner's statistics (they have no effect on the search graph), we retracted this number from the statistics of the BLP planner.

Num cells	example2	example6	example11	example12
Ours	140	218	312	389
Octree	> 500	> 2000	> 5000	> 5000

Table 2.3: Comparison with the Octree method

For examples 2, 6, 11, and 12, we compared the number of cells generated by our planner with the number of cells that a similar planner using the octree decomposition technique (see Subsection 2.2.2) would generate. Based on partial implementation, we obtained a conservative estimate (i.e., lower bound) of the number of cells generated by the octree decomposition technique. The data is reported in Table 3. Example 11 was extracted from [Avanaim, Boissonnat and Faverjon, 1988]. According to this paper, the planner described in [Faverjon, 1984], which uses the octree decomposition technique, generates more than 10,000 cells.

Therefore, the above results demonstrate that our algorithms bring major improvements to the approximate cell decomposition method.

In addition to the previous experiments, we tried to characterize empirically the efficiency of the planner as a function of the "complexity" of the workspace – i.e., the number of edges of the obstacles – and the "sparsity" of the workspace – i.e., the distance between obstacles. (The "clutteredness" of the workspace is the inverse of the sparsity.) To that purpose we ran our planner on several hundred problems using randomly generated environments with pre-specified complexity and sparsity. Figure 2.16 shows twelve of these examples. The vertical axis points up toward higher complexity and the horizontal axis points right toward smaller sparsity (i.e., greater clutteredness). (The two obstacles in the workspace are enclosed within two rectangular boxes. The sparsity is measured as the distance between the vertical sides of these two boxes.) For each complexity and sparsity, we have generated and run tens of examples. The mean value of the running time obtained for some series of examples are depicted in Figure 2.17. Figure 2.17a corresponds to the first column of Figure 2.16, Figure 2.17b to the third column, Figure 2.17c to the first row, and Figure 2.17d to the third row. Figures 2.17a and 2.17b show that, within the range
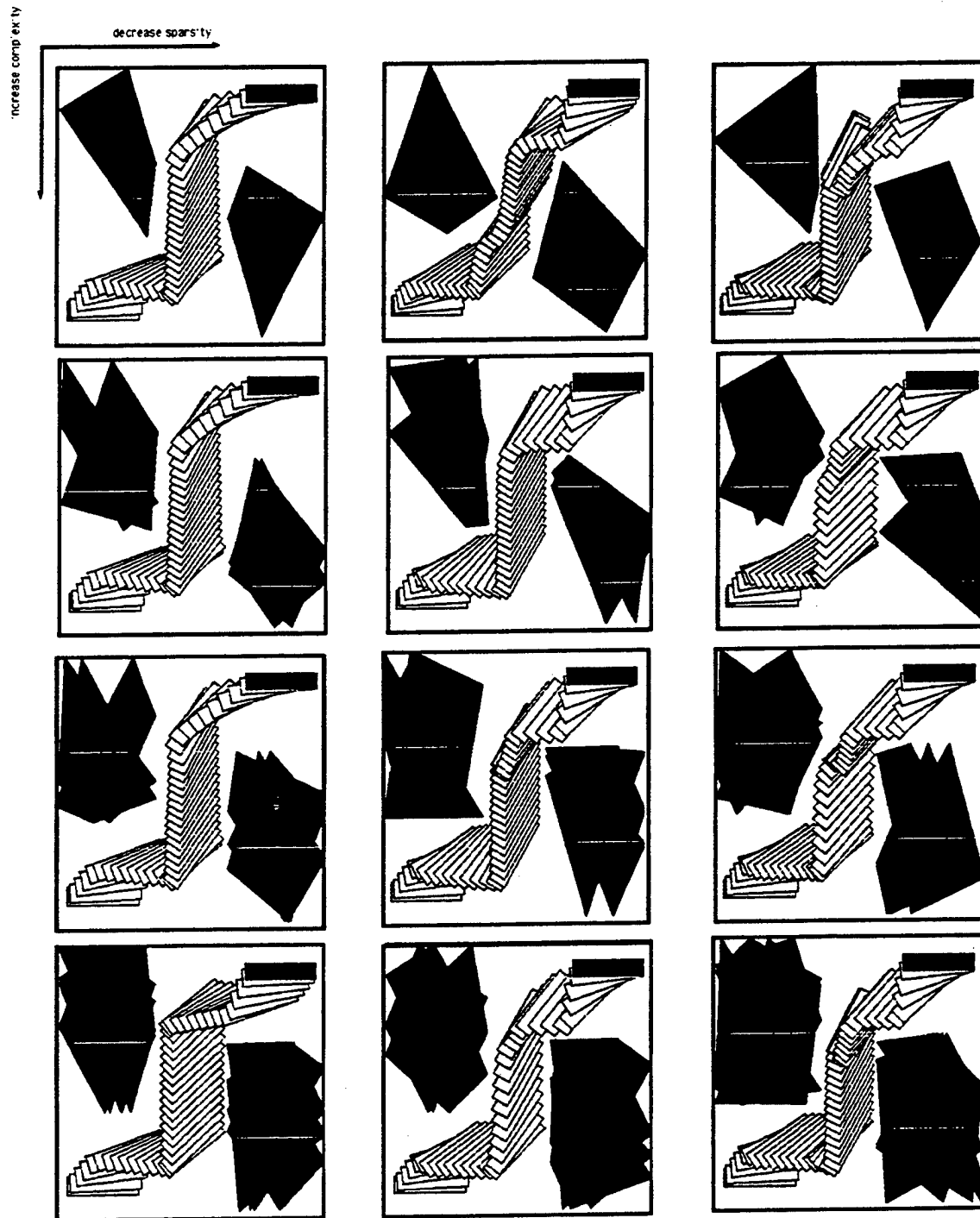
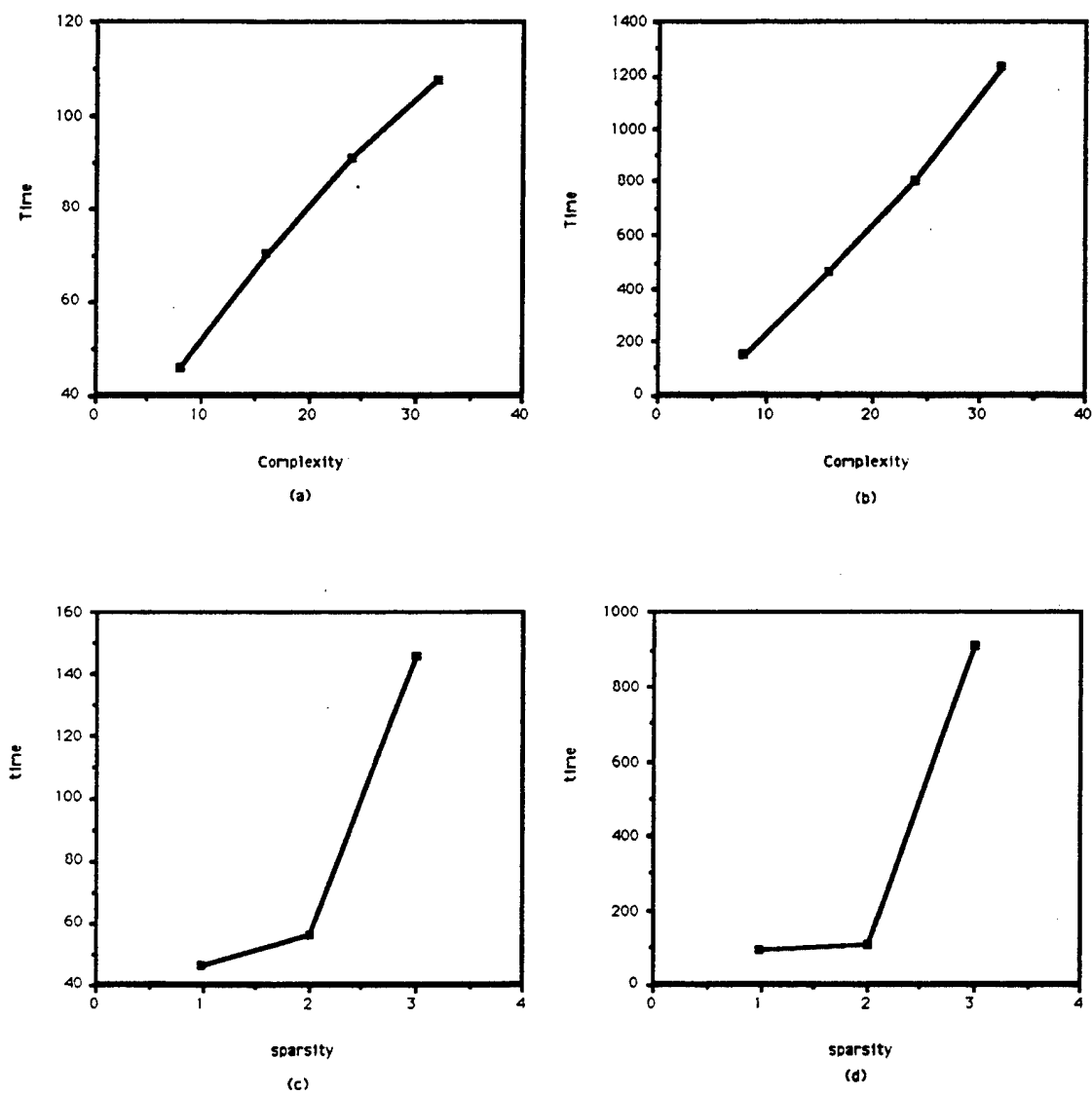Figure 2.16: Randomly generated examples

Figure 2.17: The variation of the running time of the planner as a function of the complexity (**a** and **b**) and the sparsity (**c** and **d**) of the workspace

of our experiments, the running time varies linearly with the complexity. Figures 2.17c and 2.17d show that above some clutteredness, the running time increases very quickly. This results from the fact that the planner has to generate many small cells before it has any chance to find an E-channel.

2.5 Limitation

There are two limitations related to the algorithms developed in this chapter. The first limitation is due to the choice of using rectangloid cells rather than cells with more general shapes in the constraint approximation algorithms. Rectangloid cells, although simple, impose restrictions on how accurately we can approximate the free configuration space of a robot with a given number of cells. Cells with simpler shapes have the advantage of simpler procedures for decomposition, more convenient ways for establishing connectivity between cells, and easier methods for extracting a path within a cell. On the other hand, cells with more complex shapes allow more accurate approximation of the free configuration space. When the space is uncluttered, cells with simpler shapes have advantage over cells with more complex shapes, while the converse is true when the space becomes more cluttered. An example is given in Figure 2.18 comparing trapezoidal cells with rectangular cells. When the space is less cluttered as in Figure 2.18a, rectangular cells appear to be superior. But when the space becomes more cluttered as in Figure 2.18b, trapezoidal cells appear to be superior.

This limitation can be overcome by allowing cells with different shapes to be used in a decomposition. Initially, the space is decomposed into cells with simpler shapes. But as the space is decomposed further, we can use cells with more complex shapes.

The second limitation is due to the absence of approximation in the workspace of the robot. In the constraint approximation algorithm, all approximation is done in the configuration space. Approximation in the workspace may reduce the computational cost of generating and approximating configuration space obstacles. However, when the initial approximation in the workspace leads to planning failure, the planner gets
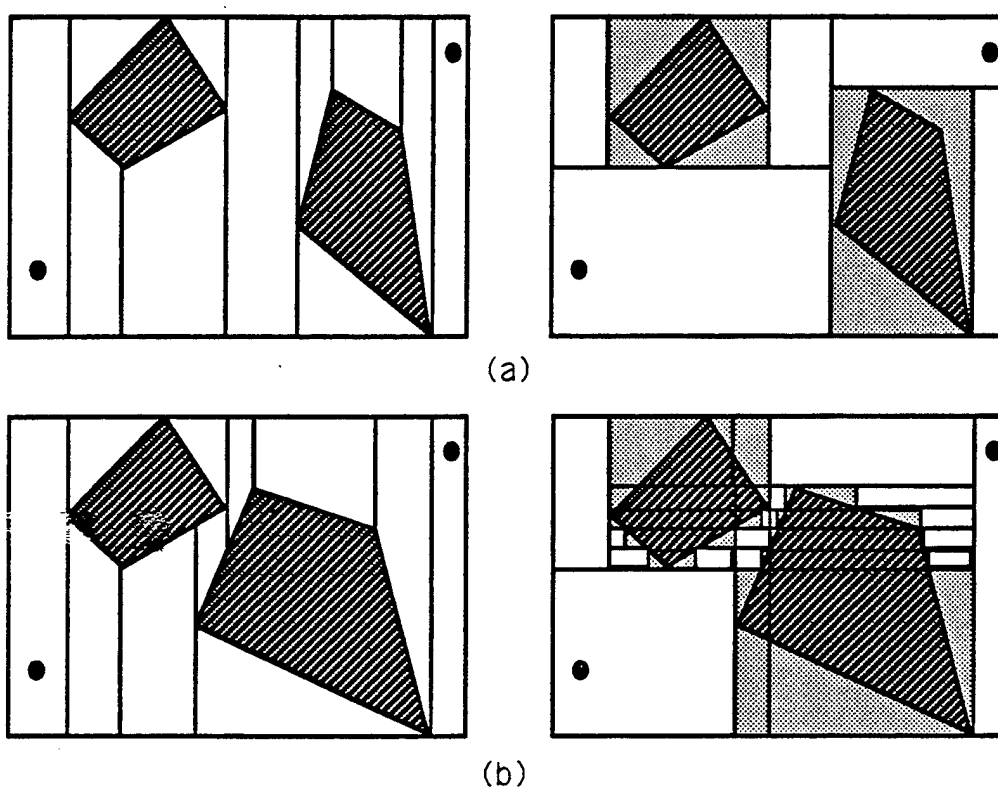
(a)

(b)

Figure 2.18: Trapezoidal cells vs. rectangular cells

no direct information as to which part of the workspace obstacles should be refined. Thus it creates difficulty in the refinement process. The impact of this limitation is that when a large portion of a complex workspace is irrelevant to finding a path, our algorithm would not be very efficient. For example, when planning a path in a room within a complex office building, our algorithm would waste a lot of time computing and approximating C-obstacles in the irrelevant part of the building.

A possible remedy for this problem is to allow the planner to choose a "scope", i.e. a region within the given workspace. All obstacles outside the chosen scope are ignored. The scope can be enlarged when the planner fails to find a path within the current scope. In the case of the previous example, the scope can be initially chosen to be the room of interest. In this way the planner will not waste time in constructing and approximating C-obstacles that are outside of the room.

2.6 Summary

In this chapter we have developed new heuristic algorithms for path planning based on the hierarchical approximate cell decomposition approach:

- *Constraint reformulation algorithms* for approximate cell decomposition. Experimental results show that these algorithms produce much better decomposition of MIXED cells than previous techniques.

- *Hierarchical search algorithms* with error detection and recovery mechanism. These algorithms allow the planner to record the conditions of past mistakes so that it does not repeat them.

We have implemented these algorithms in a path planner, with which we have conducted a variety of experiments. These experiments show that our planner is significantly (approximately 10 times) faster than previous planners based on the same general approach.

Chapter 3

Sequential Pipe Routing

In this chapter and the next, we explore the problem decomposition approach for generating problem abstractions. We conduct this exploration in the context of pipe routing problems. Pipe routing is chosen rather than multi-robot motion planning as pipe routing provides a much richer domain for studying subproblem interactions. Indeed, most realistic multi-robot motion planning problems involve only a small number of robots (typically less than ten). Furthermore, these robots are distributed over a large workspace such that even a smaller number of them are interacting with each other. On the other hand, many practical pipe routing problems involve dozens, or more, of pipes, with many pipes traversing the same region in the workspace so that their interactions are important.

Recall from Chapter 1 that routing a pipe is equivalent to planning the path of a ball, and that routing multiple pipes can be regarded as a multi-robot path planning problem. In our approach, the problem of routing many pipes is decomposed into a set of subproblems, each consisting of routing one pipe. Since the pipe routes share the same physical workspace, the subproblems interact. The key issue to be addressed is how to handle this interaction. We propose and investigate two ways of dealing with subproblem interaction: *sequential routing* with selective backtracking and *parallel routing*.

This chapter investigates the sequential routing approach with selective backtracking. In this approach the routes for the pipes are generated sequentially. The route to be generated for each pipe is constrained by the pipes that have been routed previously, and will itself constrain the other pipes that have not been routed yet. Backtracking occurs when we fail to find a route for a pipe due to routes that have been previously generated. In case of failure, the backtracking algorithm must be able to discover the cause of failure and recover from it. The main concern here is how to backtrack intelligently (selective backtracking) so that the number of backtracking operations is minimized.

The sequential approach with appropriate backtracking is efficient when the interactions among the pipes are "weak". When this is not the case, backtracking may become too frequent, independently of the qualities of the backtracking algorithm. This need for backtracking can be reduced by considering the interactions among pipes earlier, i.e. before committing pipes to specific routes. In Chapter 4 we will develop such a lesser-commitment approach, which we call parallel pipe routing.

This chapter is organized as follows: In Section 3.1, we specify the pipe routing problem that is the subject of study for this chapter and the next. In Section 3.2, we discuss the analogy between pipe routing and motion planning, and we present a spectrum of possible approaches to solve pipe routing problems. In Section 3.3, we outline the sequential routing algorithm. In Sections 3.4 and 3.5, we describe an approximate cell decomposition technique for generating pipe routes. In Section 3.6, we address the issue of backtracking and describe a selective backtracking algorithm. In Section 3.7, we provide experimental results obtained with our implemented sequential router. In Section 3.8, we discuss the limitations to the algorithms developed in this chapter. In Section 3.9, we summarize the key ideas.

3.1 The Pipe Routing Problem

Let $\mathcal{W}$ be a bounded and connected subset of $\mathbf{R}^N$, with $N = 2$ or 3, representing the pipe workspace. A **pipe specification** $\mathbf{P}_k$ is a triple (T_k^1, T_k^2, r_k), where T_k^1 and

T_k^2 are the two distinct points in $\mathcal{W}$ representing the **terminals** that the pipe should connect, and r_k is the radius of the pipe. A **route** for $\mathbf{P}_k$ is the region $\mathcal{R}_k$ in $\mathcal{W}$ that is swept out by a disc (in two-dimensional workspace) or a ball (in three-dimensional workspace) of radius r_k when its center moves along a curvilinear line L_k connecting T_k^1 and T_k^2. This line is called the **path** of the pipe route.
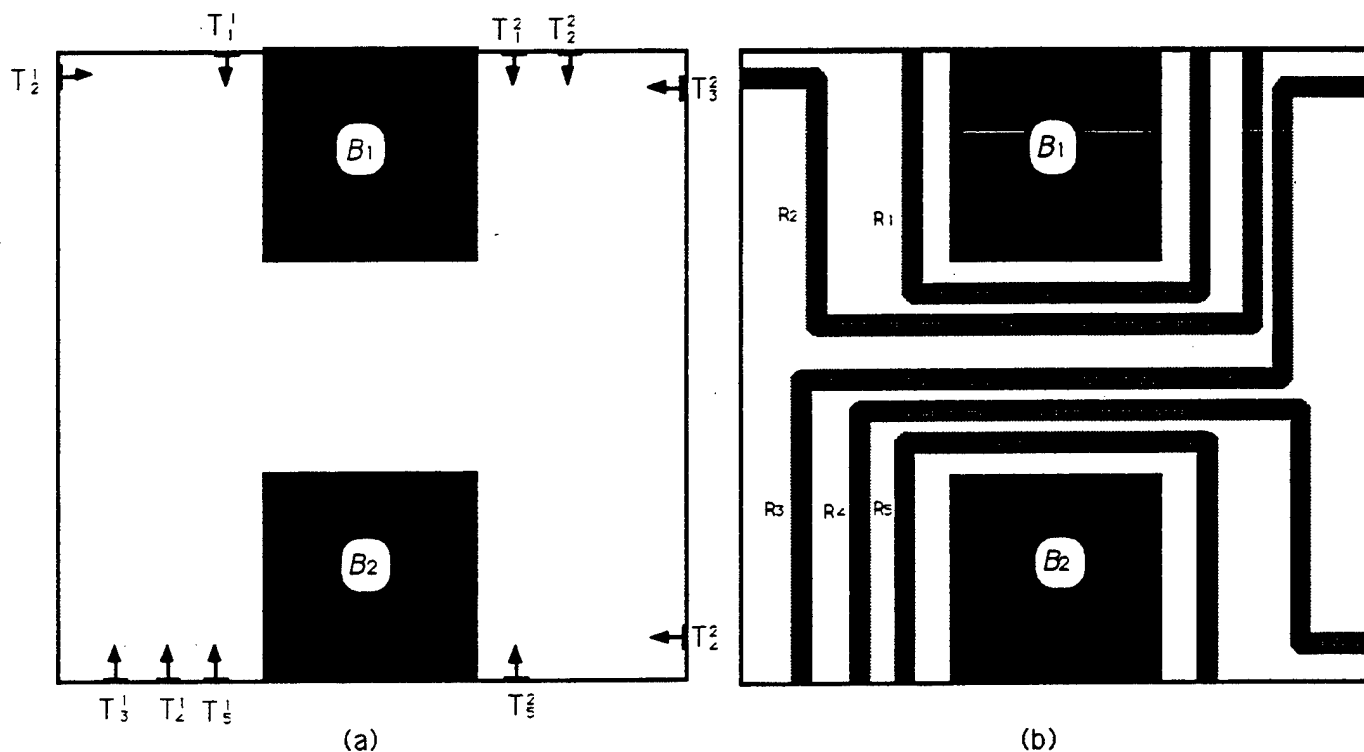

Figure 3.1: An example of the Basic Pipe Routing (BPR) problem

We formulate the **Basic Pipe Routing (BPR)** problem as follows:

Given a collection of obstacles $\mathcal{B}_i$ $(i = 1, ..., n)$ in $\mathcal{W}$ and a set of pipe specifications $\mathbf{P}_k$ $(k = 1, ..., p)$, compute routes $\mathcal{R}_k \subset \mathcal{W}$ $(k = 1, ..., p)$, so that:

- no route intersects an obstacle, i.e. : $\forall i \in [1, n], \forall k \in [1, p] : \mathcal{B}_i \bigcap \mathcal{R}_k = \emptyset$;

- no two routes intersect each other, i.e. : $\forall k, k' \in [1, p], k \neq k' : \mathcal{R}_k \bigcap \mathcal{R}_{k'} = \emptyset$.

(Since the routes must lie in $\mathcal{W}$, the boundary of $\mathcal{W}$ is treated as the boundary of an additional obstacle enclosing the workspace. We denote this "obstacle" by $\mathcal{B}_0$.)

Figure 3.1 shows an example of the BPR problem in a two-dimensional rectangular workspace. In this example, there are two polygonal obstacles $\mathcal{B}_1$ and $\mathcal{B}_2$ (shown as dark) and five pipe specifications (Figure 3.1a). A solution to this problem is shown in Figure 3.1b.

In this chapter and the next, we make the following assumptions:

- Both the workspace $\mathcal{W}$ and the obstacles $\mathcal{B}_i$ are modeled as polygons or polyhedra.

- All the terminals are located in the boundary of the region $\mathcal{E} = \mathcal{W} \setminus \cup_i \mathcal{B}_i$. At every terminal the path of the corresponding pipe is perpendicular to the boundary of $\mathcal{E}$.

- The path of every pipe is a sequence of straight and circular segments. All the circular segments in the same path L_k have the same radius ρ_k specified as an additional parameter in the pipe specification.

- A Cartesian coordinate system $\mathcal{F}_\mathcal{W}$ is embedded in $\mathcal{W}$. Every straight segment in the path of a pipe is parallel to one of the axes of $\mathcal{F}_\mathcal{W}$ and (in the three-dimensional case) every circular segment is parallel to one of the planes determined by two axes of $\mathcal{F}_\mathcal{W}$.

These assumptions are all reasonable in practice. In most cases, $\mathcal{W}$ is a parallelepiped bounded by "walls". The axes of $\mathcal{F}_\mathcal{W}$ are selected to be parallel to the edges of this parallelepiped. "Good" pipe design tends to avoid slanted pipes unless it is absolutely necessary.

3.2 Approaches to Pipe Routing

3.2.1 Pipe Routing and Multi-Robot Path Planning

The pipe routing problem and the robot path planning problem are very similar. Indeed, we can regard a pipe route as the trace left behind by a rigid object moving in

the pipe's workspace. Using this metaphor, we transform the pipe routing problem into the problem of planning the motion of a collection of robots. In fact, there is a spectrum of techniques (shown in Figure 3.2) for planning the motions of multiple robots, ranging from treating all robots as a single composite robot (centralized planning on the extreme right) to planning a path for one robot at a time independently of the other robots (independent planning on the extreme left). Except for the pure centralized approach, all the other approaches decompose the problem of finding paths for multiple robots into a set of subproblems, each consisting of finding a path for one robot. These are often referred to as the decentralized approaches. Except for the pure independent approach, all the decentralized approaches deal with the interactions among the subproblems to some extent. They differ in how and when the interactions among the subproblems are considered. As we move from the right to the left of the spectrum, we gain efficiency but lose completeness. The centralized planning approach, the prioritized planning approach, and the velocity tuning approach have all been proposed and applied previously in [Schwartz and Sharir, 1983c], [Erdmann and Lozano-Pérez, 1986], and [Kant and Zucker, 1986b], respectively. These three approaches have been shown to work for a small number of robots. The sequential approach with backtracking and the lesser-commitment approach are the subjects of this chapter and the next. In this section we provide a brief overview of some of these approaches.
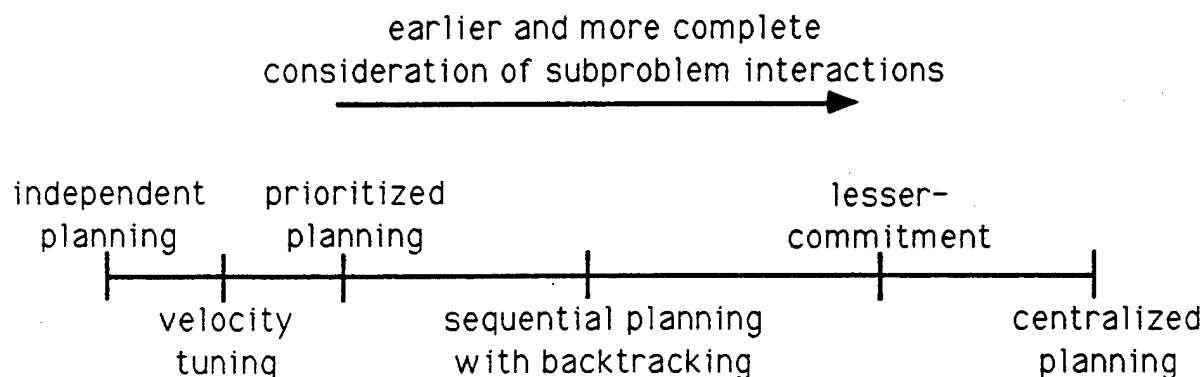

Figure 3.2: A spectrum of different approaches to multiple-robot path planning

3.2.2 Centralized Approach

A conceptually simple approach to multi-robot path planning is to treat the individual robots as the components of a single, multi-bodied robot $\mathcal{A} = \{\mathcal{A}_1, ..., \mathcal{A}_p\}$. The configuration space of $\mathcal{A}$ is $\mathcal{C} = \mathcal{C}_1 \times ... \times \mathcal{C}_p$, with $\mathcal{C}_k$ ($k \in [1, p]$) denoting the configuration space of $\mathcal{A}_k$. The obstacles, including those due to the interactions among the robots, are mapped into $\mathcal{C}$ and a free path is planned among the C-obstacles. Such a path determines a set of p *coordinated* free paths for the robots $\mathcal{A}_k$ in their respective configuration spaces. In this way some basic motion planning methods apply to multi-robot path planning.

This approach to multi-robot path planning is called the *centralized planning* approach. Although it solves the problem in theory, it suffers, however, from a major drawback: the dimension m of $\mathcal{C}$ is equal to p times the dimension of each individual space $\mathcal{C}_k$ (assuming all these spaces have the same dimension). Since finding a free path in a configuration space of dimension m requires exponential time in m, the approach tends to be impractical, especially when p is large (which is often the case in pipe routing). In the particular case of the approximate cell decomposition method, it yields a number of cells that is exponential in the dimension of $\mathcal{C}$.

3.2.3 Decentralized Approaches

The decentralized approaches cope with the problem of high dimensionality of the configuration space by decomposing the multi-robot path planning problem into a set of subproblems, each consisting of planning a path for one robot.

The independent planning approach is the extreme of the decentralized approaches. It finds a path for each robot independent of the other robots. Of course this simple-minded approach would not work in general because these robots share the same physical space and hence their paths may interact. Thus the paths found independently may not be compatible with each other.

The velocity tuning approach , proposed in [Kant and Zucker, 1986b], deals with the interactions among the robots by tuning their velocities along their chosen paths to avoid collision. It first plans a path for every robot independent of the other robots. Two robots may collide if their paths are closer than the sum of their radii at some points along the paths. In this case, the velocities of these two robots are tuned so that one robot will pass this "critical region" before the other robot arrives there. It is possible that the paths generated in the first phase cannot be "tuned" to avoid collision (consider the example where two robots are heading toward each other along a narrow corridor). In this case, this approach will fail.

The prioritized planning approach , described in [Erdmann and Lozano-Pérez, 1986], deals with the interaction problem by planning for the robots in some pre-specified order (based on a priority scheme). It plans a path for one robot at a time, with those robots whose motions have already been planned being considered as moving obstacles[1]. This approach makes two implicit assumptions: the paths of the robots can be planned in some sequential order; and this order is known in advance. Both of these two assumptions may turn out to be false. There are examples where the paths of the robots cannot be planned sequentially (consider the example where two robots want to exchange their positions in a narrow corridor). Even when they can be planned in sequence, the correct ordering may not be known in advance. If either of these two assumptions is false, this approach will fail.

The sequential approach with backtracking is an extension of the prioritized approach. Like the prioritized approach, the sequential approach with backtracking generates a path for one robot at a time. But when it cannot find a path for a robot, it changes the path(s) generated previously for some robot(s) in order to free up space for the current robot. The key issues here are deciding which paths to change and how to change them. This is the subject of investigation of this chapter (sequential routing).

---

[1]These moving obstacles (as well as the workspace obstacles) are mapped as static regions in the configuration-time space of the robot, which is defined as the product of the configuration space and the time axis. A path is constructed as a time-monotonic curve in this space

The lesser-commitment approach is an attempt to reduce the need for back-tracking which is required by the sequential approach when the interactions among the robots are strong. The lesser-commitment approach first chooses a channel for each robot to go through without specifying a path in this channel. It considers the interactions among the robots only when their channels overlap. Interaction among robots leads to the refinement of the corresponding channels. After the channels have been refined, a specific path is constructed in each of them. This lesser-commitment approach will be the subject of study of Chapter 4 (parallel routing).

3.3   Sequential Routing Algorithm

Our sequential pipe routing algorithm is an adaptation of the approximate cell decomposition method developed in robot motion planning. It considers the pipes $\mathbf{P}_k$, $k \in [1, p]$, in sequence. For each pipe $\mathbf{P}_k = (T_k^1, T_k^2, r_k)$, it plans a collision-free path for a ball of radius r_k from T_k^1 to T_k^2 among the obstacles $\mathcal{B}_0, ..., \mathcal{B}_n$ and the pipe routes $\mathcal{R}_1, ..., \mathcal{R}_{k-1}$ already constructed. If such a path L_k is found, it defines a route $\mathcal{R}_k$. Otherwise, the algorithm must backtrack, i.e. it must change some of the routes $\mathcal{R}_1, ..., \mathcal{R}_{k-1}$ in order to make room for $\mathcal{R}_k$. As mentioned in the previous section, this corresponds to applying a prioritized planning approach with backtracking. A simple, but general prioritizing heuristics to reduce backtracking is to route bigger pipes before smaller ones.

We let $\mathcal{A}_k$ designate the "robot" tracing out the route $\mathcal{R}_k$ and $\mathcal{C}_k$ denote its configuration space. The algorithm that is executed at each iteration in order to compute the path L_k of $\mathcal{R}_k$ consists of the following three steps:

1. Compute the C-obstacles due to the obstacles $\mathcal{B}_0, ..., \mathcal{B}_n$ and the existing routes $\mathcal{R}_1, ..., \mathcal{R}_{k-1}$ by growing them by the radius r_k of $\mathbf{P}_k$. Approximate the free space in $\mathcal{C}_k$ as a conservative collection of (EMPTY) rectangloid cells.

2. Construct the connectivity graph of these cells and search this graph for a channel, i.e. a sequence of cells such that the boundary of the first (resp. the last) cell contains T_k^1 (resp. T_k^2) and any two successive cells are adjacent.

3. If the search terminates successfully, extract a path L_k from the channel. Otherwise return failure.

In order to facilitate the subsequent extraction of paths, the rectangloid cells generated at Step 1 have all their sides parallel to axes of $\mathcal{F}_W$. Two cells are adjacent if the intersection of their boundaries is a segment of non-zero length (in $\mathbf{R}^2$) or a rectangle of non-zero area (in $\mathbf{R}^3$).
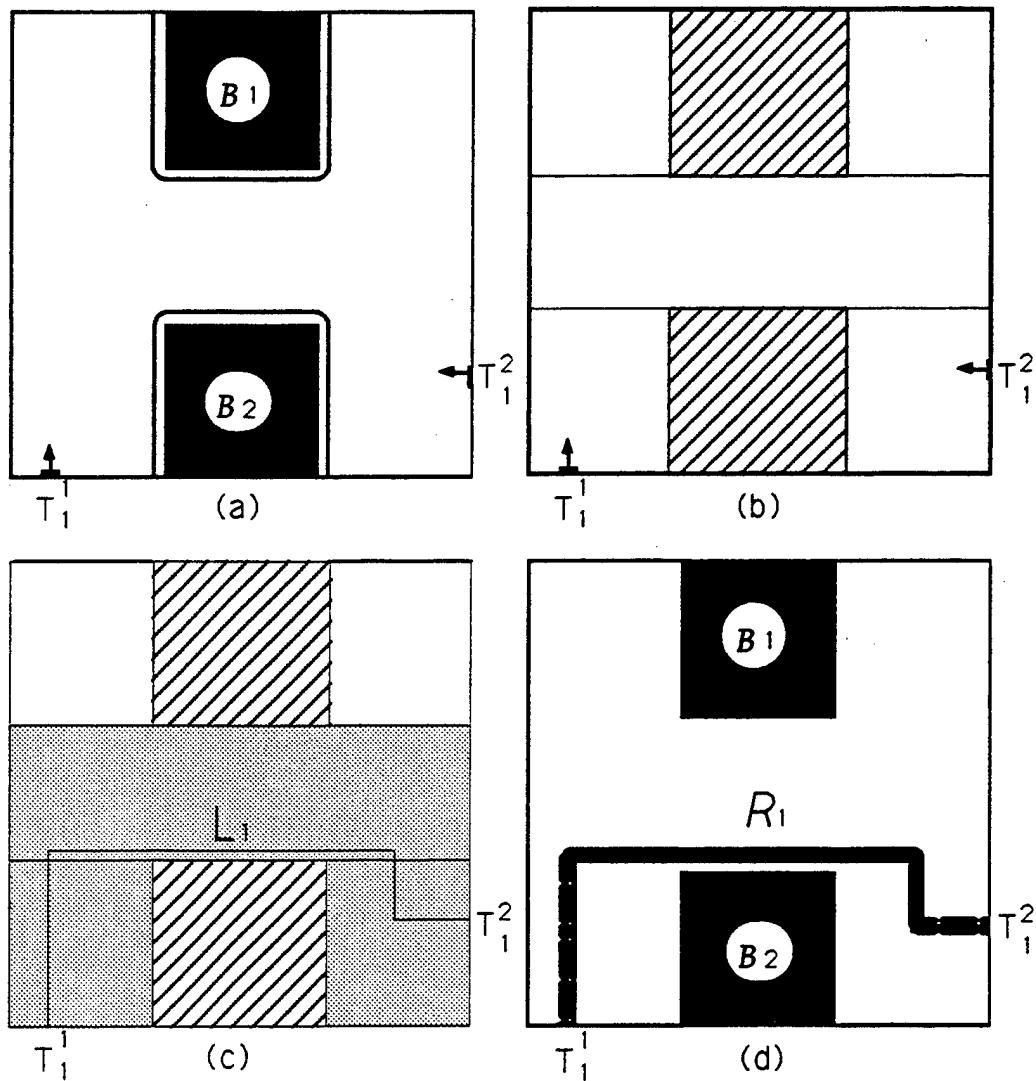

Figure 3.3: A pipe routing example

Figure 3.3 illustrates the operations performed by this algorithm in a two-dimensional workspace for a single pipe $\mathbf{P}_1$. Figure 3.3a shows the construction of the C-obstacles due to $\mathcal{B}_1$ and $\mathcal{B}_2$ in the configuration space $\mathcal{C}_1$. Figure 3.3b displays an approximation of the free subspace in $\mathcal{C}_1$ in the form of a collection of rectangular cells. Figure 3.3c shows both a channel between the two terminals T_1^1 and T_1^2 and a path L_1 in this channel. Figure 3.3d presents the corresponding route $\mathcal{R}_1$ in the workspace.

The operations carried out by the above algorithm are described in more detail in the following sections. We also present the backtracking mechanism that is activated when the path planning algorithm fails to construct a path L_k.

3.4 Cell Decomposition

The first step of the above algorithm generates an explicit conservative representation of the space that is available for the pipe $\mathbf{P}_k$ — i.e. the free space in $\mathcal{C}_k$ — in the form of a collection of rectangloid cells. To this purpose, it first maps the obstacles $\mathcal{B}_i$ ($i = 0$ to n) and the routes $\mathcal{R}_j$ ($j = 1$ to $k - 1$) to C-obstacles in $\mathcal{C}_k$. We denote these C-obstacles by $\mathcal{CB}_i$ and $\mathcal{CR}_j$, respectively. Next, it computes a bounding approximation of these C-obstacles in the form of a collection of rectangloids whose edges are parallel to $\mathcal{F}_W$'s axes. The complement in $\mathcal{C}_k$ of all these rectangloids is easily computed in the form of a collection of rectangloids also oriented along $\mathcal{F}_W$'s axes. This second collection of rectangloids are the cells of the conservative representation of the free space in $\mathcal{C}_k$.

3.4.1 The Two-Dimensional Case

$\mathcal{CB}_i$, for $i = 0$ to n, is obtained by growing $\mathcal{B}_i$ isotropically by the radius r_k of the pipe to be routed. Since we model $\mathcal{B}_i$ as a polygon, $\mathcal{CB}_i$ is a generalized polygon whose boundary is a sequence of straight and circular edges as shown in Figure 3.4. Each straight edge of $\mathcal{CB}_i$ is the locus of the center of the disc $\mathcal{A}_k$ when it moves with its boundary sliding in contact along an edge of $\mathcal{B}_i$. It is obtained by translating the

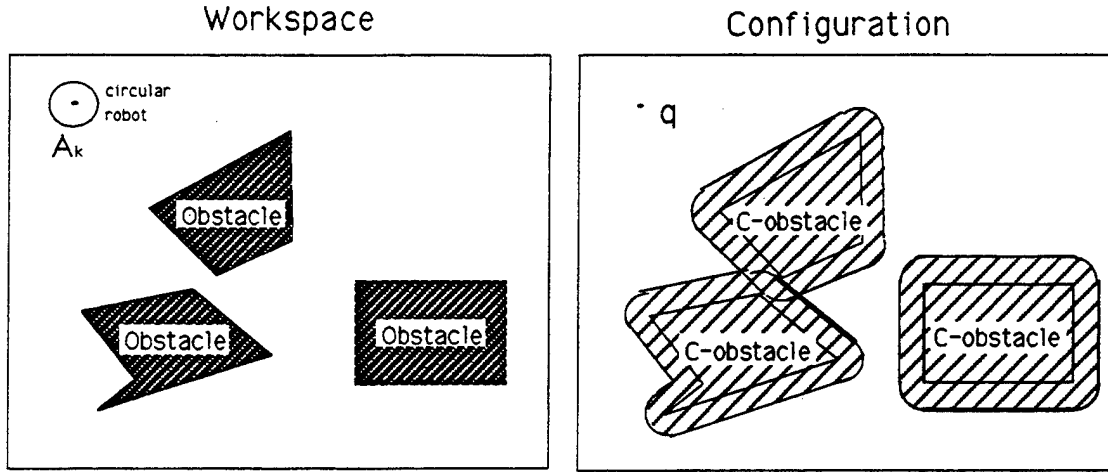Workspace   Configuration

Figure 3.4: C-obstacle of a polygon

corresponding edge of $\mathcal{B}_i$ by r_k along its outward normal direction. Each circular edge is the locus of the center of $\mathcal{A}_k$ while it moves with its boundary being in contact with a convex vertex V of $\mathcal{B}_i$. It is a circular arc of radius r_k centered at V. Let n be the number of vertices of $\mathcal{B}_i$. Algorithms compute $\mathcal{CB}_i$ in $O(n)$ time, if $\mathcal{B}_i$ is convex, and in $O(n^2 \log n)$ time, otherwise [Latombe, 1991]. In the second case, if the intersection of $\mathcal{B}_i$ with any disc of radius r_k contains a small number of vertices (that is then considered to be constant), a reasonable situation where $\mathcal{B}_i$ is said to have *bounded local complexity* [Sifrony and Sharir, 1987], the time complexity reduces to $O(n \log n)$.

$\mathcal{CR}_j$, for $j = 1$ to $k-1$, is also obtained by growing $\mathcal{R}_j$ isotropically by the radius r_k. It has the form of a route of radius $r_j + r_k$ following the same path L_j as $\mathcal{R}_j$. The shape of an elbow of $\mathcal{CR}_j$ corresponding to a circular segment of L_j (of radius ρ_j) depends on the relative values of r_j, r_k, and ρ_j. If $\rho_j - (r_j + r_k) > 0$, then the elbow of $\mathcal{CR}_j$ is bounded by two circular arcs of respective radii $\rho_j + r_j + r_k$ and $\rho_j - r_j - r_k$ (see Figure 3.5a). Otherwise, it is bounded by a circular arc of radius $\rho_j + r_j + r_k$ on one side and by a right corner on the other side (Figure 3.5b).

We compute a rectangular approximation of $\mathcal{CB}_i$ by first constructing the bounding box of $\mathcal{CB}_i$ whose edges are parallel to $\mathcal{F}_W$'s axes. We next cut this box into slices
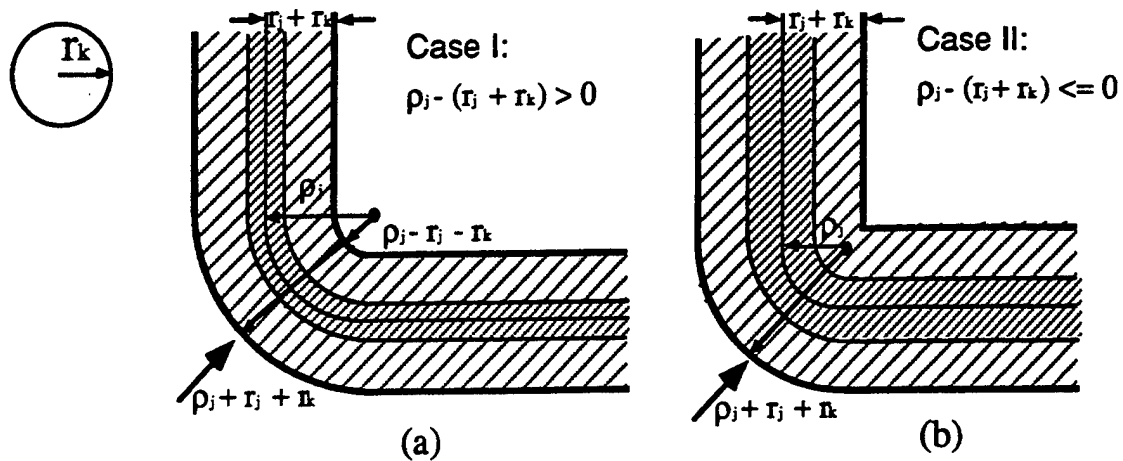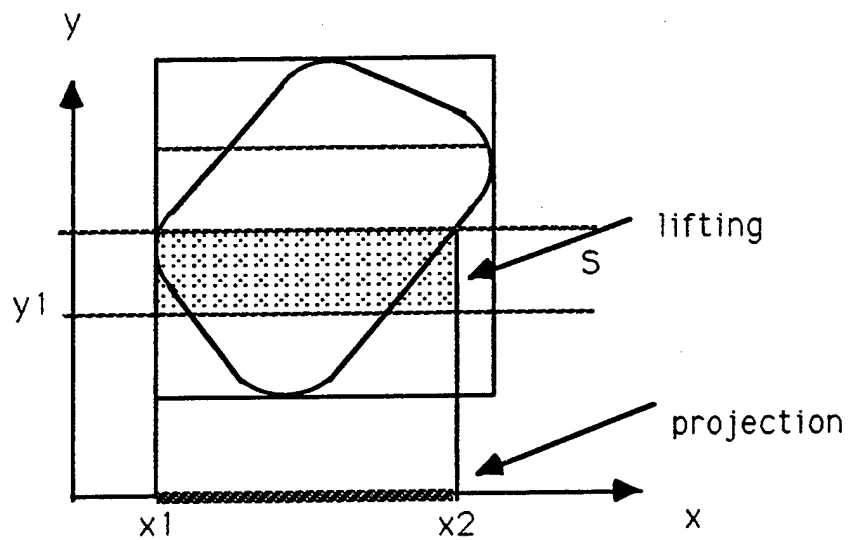
Figure 3.5: C-obstacle due a pipe elbow

Figure 3.6: Computation of the bounding approximation of a C-obstacle

parallel to its shortest side at a specified resolution (which determines the precision of the approximation). Assume that these slices are parallel to the x-axis. For each slice S, we compute the projection of the intersection of $\mathcal{CB}_i$ and S on the x-axis, and we lift this projection back into the slice, thus obtaining a rectangloid approximation of $\mathcal{CB}_i$ within S (see Figure 3.6).

Every grown route $\mathcal{CR}_j$ is treated in a similar fashion. However, we take advantage of the special shape of the routes to simplify this process. In particular, since the straight segments of a route are rectangles oriented along $\mathcal{F}_W$'s axes, they do not have to be approximated.

3.4.2   The Three-Dimensional Case

The isotropic growth by r_k of a polyhedral obstacle $\mathcal{B}_i$ yields a generalized polyhedral C-obstacle $\mathcal{CB}_i$, i.e. a volume whose boundary is made of pieces of planar, cylindrical, and spherical surfaces. Each planar face corresponds to a face F of $\mathcal{B}_i$, and is obtained by translating F by r_k along its outward normal direction. Each cylindrical face corresponds to a convex edge E of $\mathcal{B}_i$, and is part of the surface of a cylinder having E as its midline and r_k as its radius. Each spherical face corresponds to a convex vertex V of $\mathcal{B}_i$, and is part of the sphere of radius r_k centered at V.

In order to compute a bounding approximation of $\mathcal{CB}_i$, we apply a recursive project-and-lift technique [Zhu and Latombe, 1991a], which we sketch below. We assume that the longest dimension of the bounding box of $\mathcal{CB}_i$ is along $\mathcal{F}_W$'s z-axis. We then slice the z-axis into intervals at some pre-specified resolution, and we compute the projection $\pi_{xy}(\mathcal{CB}_i \cap [z_1, z_2])$ of the subset of $\mathcal{CB}_i$ contained in every slice $[z_1, z_2]$ into the xy-plane. Next, we proceed recursively by slicing, say, the y-axis and projecting the subset of $\pi_{xy}(\mathcal{CB}_i \cap [z_1, z_2])$ contained in every slice $[y_1, y_2]$ into the x-axis. The latter projection consists of one or several intervals. We lift each such interval $[x_1, x_2]$ up to a rectangloid $[x_1, x_2] \times [y_1, y_2] \times [z_1, z_2]$ in $\mathcal{C}_k$.

We could compute the projection $\pi_{xy}(\mathcal{CB}_i \cap [z_1, z_2])$ from the explicit representation of $\mathcal{CB}_i$'s boundary. Although the construction of this representation is not really

difficult, it is not needed as shown below (if we assume that $\mathcal{B}_i$ is homeomorphic to a sphere, hence it does not have any hole.) For every face F of $\mathcal{B}_i$, we let $C\mathcal{F}$ denote the face F translated by r_k along its outward normal. For every convex edge E of $\mathcal{B}_i$, we let $C\mathcal{E}$ denote the truncated cylinder of radius r_k having E as its midline. For every convex vertex V of $\mathcal{B}_i$, we let $C\mathcal{V}$ denote the sphere of radius r_k centered at V. Since both the cylinders $C\mathcal{E}$ and the spheres $C\mathcal{V}$ are entirely contained in $C\mathcal{B}_i$, the projection $\pi_{xy}(C\mathcal{B}_i \cap [z_1, z_2])$ is obtained by projecting the subset of every element $C\mathcal{F}$, $C\mathcal{E}$, and $C\mathcal{V}$ contained in $[z_1, z_2]$ into the xy-plane, and computing the external boundary of the union of all the projections. These computations are straightforward (though, in the case of a cylinder $C\mathcal{E}$, several different cases have to be carefully considered). If n is the number of vertices of $\mathcal{B}_i$, the overall computation takes $O(n^2 \log n)$ time in the worst case, using a sweep-line technique [Preparata and Shamos, 1985] to trace out the external boundary of $\pi_{xy}(C\mathcal{B}_i \cap [z_1, z_2])$. This boundary consists of straight, circular, and elliptical edges.

The isotropic growth of a route R_j by r_k yields a route of radius $r_j + r_k$ following the same path L_j as R_j. The straight segments and the elbows of the grown route are treated separately. Each straight segment is parallel to a plane of $\mathcal{F}_\mathcal{W}$, say the xy-plane. It is sliced into intervals along the z-axis. Each slice projects into the xy-plane as a rectangle which is lifted up into a rectangloid. Each elbow of the grown route is also parallel to a plane, say the xy-plane. Its cross-section by a plane at any constant z has one of the two forms shown in Figure 3.5. Hence, each elbow can be sliced into intervals along the z-axis, with each slice projecting into the xy-plane as a two-dimensional route. Each projection is approximated as in the two-dimensional case and the resulting rectangular cells are lifted up into three-dimensional rectangloid cells.

3.4.3 Cashing in on Previous Computation

If the current route has the same radius as a previously generated route (a frequent case in practice), we can reuse the approximations of the $C\mathcal{B}_i$'s computed for planning the previous route. If two routes $\mathcal{R}_{k-1}$ and $\mathcal{R}_k$ having the same radius are planned

consecutively, we can even save more work. Indeed, most of the cell decomposition of the free space in C_{k-1} (and consequently, most of the connectivity graph) remains valid in C_k. One can proceed as follows. The decomposition built to represent the free space in C_{k-1} is copied for C_k and, since the route $\mathcal{R}_{k-1}$ is just an additional obstacle to $\mathcal{R}_k$, the cells of the copied decomposition which are intersected by the route $\mathcal{R}_{k-1}$ grown by $r_k = r_{k-1}$ (and only these cells) are refined into smaller cells. Since this can result in considerable saving of computation, it is generally desirable to plan routes with the same radius consecutively. This is compatible with the general prioritizing heuristics that suggest that bigger pipes should be routed before smaller ones.

3.5   Channel and Route Generation

Once we have decomposed the free space in C_k, we construct the **connectivity graph** CG_k representing the adjacency relation among the generated cells. The node corresponding to the cell whose boundary contains the terminal T_k^1 (resp. T_k^2) is called the **initial** (resp. **goal**) node (the initial and goal nodes coincide if the two terminals are contained in the boundary of the same cell). A channel is constructed by searching CG_k for a path connecting the initial node to the goal node.
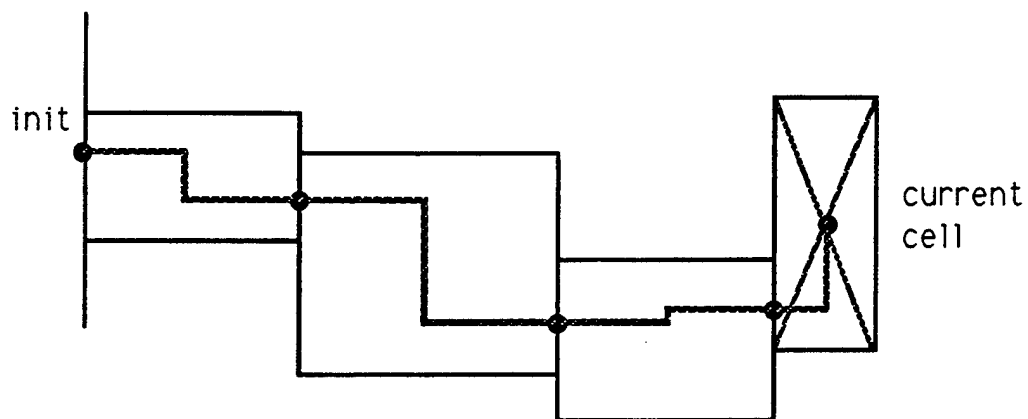

Figure 3.7: Estimation of the length of a path in a partial channel

Various techniques could be used to search CG_k. In our implementation, we use an A* algorithm [Nilsson, 1980]. The search is guided by an evaluation function

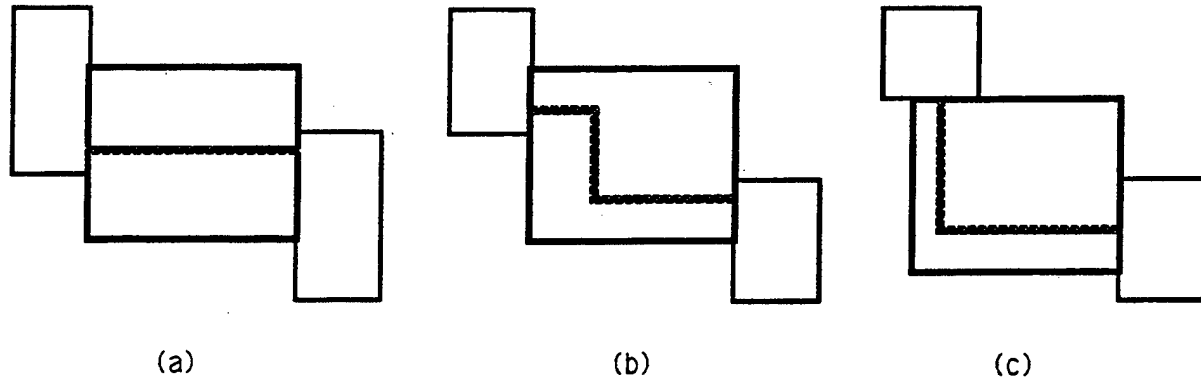(a)   (b)   (c)

Figure 3.8: Expected number of turns in a cell

$f(N) = g(N) + h(N)$ defined over the set of nodes in CG_k. If there exist one or several channels between the initial and the goal cells, the A* algorithm generates a channel that minimizes the function f evaluated at the goal node. Since it is desirable to plan paths of minimal length and minimal number of turns in order to generate satisfactory layouts, we use the evaluation function f to encode this desire.

Hence, $g(N)$ is defined as a weighted sum of the length $l(N)$ of the path L_k constructed so far, and its number of turns $n(N)$. However, since we construct L_k only after a complete channel has been generated, $l(N)$ and $n(N)$ are only estimates derived from the geometry of the partial channel. We compute $l(N)$ as the Manhattan length of the line connecting the terminal T_k^1 to the center of the last cell generated so far, and passing through the midpoint of the intersection of every two successive cells (Figure 3.7 shows this line in the two-dimensional case). We compute $n(N)$ by associating an expected number of turns in every cell of the partial channel constructed so far, except the last cell. We define the *entrance* of a cell κ as the intersection of its boundary with that of the previous cell (the entrance of the first cell is the point T_k^1) and its *exit* as the intersection of its boundary with that of the following cell (the exit of the last cell is T_k^2). In the two-dimensional case, the expected number of turns in κ is: 0, if the entrance and the exit are contained in parallel sides of the cell and their projections overlap (Figure 3.8a); 2, if the entrance and the exit are in parallel sides and their projections do not overlap (Figure 3.8b); and 1, if the entrance and

the exit are not in parallel sides (Figure 3.8c). In the three-dimensional case, the expected number of turns in κ is 0, 1, 2, or 3 depending on the relative orientation and position of the entrance and the exit of the cell.

The function $h(N)$ is simply computed as the Manhattan distance between the center of the cell corresponding to the node N and the terminal T_k^2.

Let us assume that the search of CG_k succeeds in producing a channel $(\kappa_1, ..., \kappa_s)$ of s adjacent cells. We construct a path L_k within this channel which has minimal length and minimal number of turns over all the paths lying in this channel. This is carried out in two steps. First we select a point Q_i in the exit of every cell κ_i ($i \in [1, s-1]$) along the channel, except the last one, and then we construct L_k by concatenating s subpaths successively connecting T_k^1 to Q_1 in κ_1, Q_i to Q_{i+1} in κ_{i+1}, for $i = 1, ..., s-1$, and Q_{s-1} to T_k^2 in κ_s. The choice of the points Q_i is made by first applying the following two-pass constraint propagation technique to refine the exits of the cells, and then selecting Q_i within the refined exits.

Let β_i denote the exit of the cell κ_i with $\beta_0 = T_k^1$ and $\beta_s = T_k^2$. In the first pass, for $i = 1$ to $s - 1$, we set β_i to its intersection with the "projection" of β_{i-1} into the line (two-dimensional case) or the plane (three-dimensional case) containing β_i, if this intersection is non-empty; otherwise, we leave it unchanged. The "projection" is computed differently for the two-dimensional case than for the three-dimensional case. In the two-dimensional case, let β_{i-1} be an interval parallel to the x-axis and represented by $[x_1, x_2]$. The projection is another interval parallel to the x-axis and represented by $[x_1, x_2]$ if β_i is also parallel to the x-axis (shown in Figure 3.9a). The projection is an empty set if β_i is parallel to the y-axis (shown in Figure 3.9b). In the three-dimensional case, let β_{i-1} be a rectangular region parallel to the xy-plane and represented by $[x_1, x_2, y_1, y_2]$. The projection is another rectangular region parallel to the xy-plane and represented by $[x_1, x_2, y_1, y_2]$ if β_i is also parallel to the xy-plane (shown in Figure 3.9c). The projection is a rectangular region parallel to the yz-plane and represented by $[y_1, y_2, z_1^C, z_2^C]$ if β_i is parallel to the yz-plane (shown in Figure 3.9d). The projection is a rectangular region parallel to the xz-plane and represented by $[x_1, x_2, z_1^C, z_2^C]$ if β_i is parallel to the xz-plane (shown in Figure 3.9e). z_1^C and z_2^C

are the minimum and maximum z coordinates of the cell κ_i. In the second pass, for $i = s - 1$ to 1, we set β_i to its intersection with the projection of β_{i+1}, if this intersection is non-empty. In choosing Q_i, we let $Q_0 = T_k^1$, and for $i = 1, ..., s - 1$, we set Q_i to the projection of Q_{i-1} into β_i if the projection lies within β_i; otherwise we set Q_i to the point in β_i that is closest to Q_{i-1}, subject to the constraints described below concerning the turn radius of the pipe. This selection guarantees the generation of a path with minimum length and minimum number of turns within the channel.
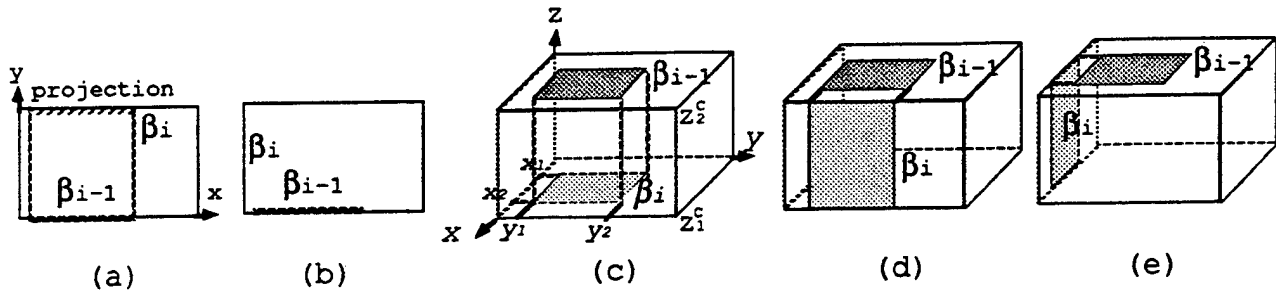
Figure 3.9: Projecting β_{i-1} into β_i.

Since $g(N)$ can only estimate the cost (weighted sum of length and number of turns) of the path going to N, the channel generated by the A* search is not guaranteed to be optimal. Consequently, the best path generated within the channel may not be optimal. In general, however, it is close to optimal.
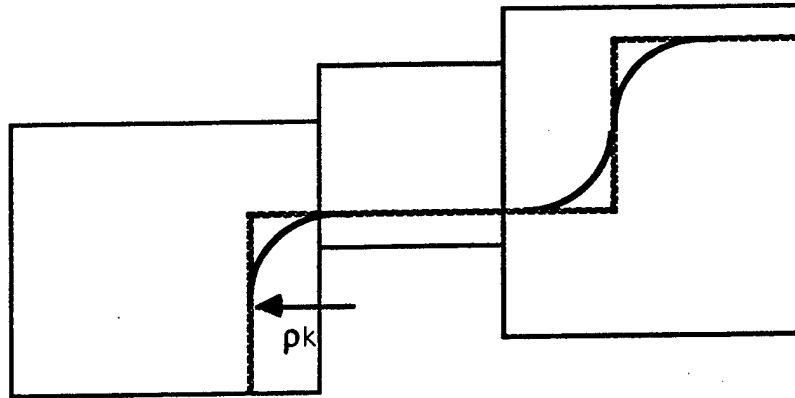
Figure 3.10: Fitting circular arcs into a path

The algorithm described above generates a pipe path L_k in the form of a polygonal line connecting two terminals. The intermediate vertices of this line must be changed into circular arcs of radius ρ_k as illustrated in Figure 3.10. It may, however, happen that this transformation is impossible (for instance, two consecutive vertices of the path are closer to each other than $2\rho_k$), or that a circular arc intersects a C-obstacle. If either of these two events happens, we can either relax the optimality requirement discussed above or resume the search of CG_k at the cell of the current channel where the problem was detected. (Remark: This algorithm is not complete. The problem of finding a path with a lower-bounded curvature radius is directly related to the path planning problem for a nonholonomic car-like robot with a limited steering angle [Laumond, 1987] [Fortune and Wilfong, 1988] [Jacobs and Canny, 1989] [Barraquand and Latombe, 1989b] [Latombe, 1991]. This problem is known to be difficult. Fortunately, in pipe layout design, the radius ρ_k is usually small, and the two events mentioned above happen relatively rarely.)

After a path L_k has been constructed, we update the layout description by adding the newly created route. This route becomes an obstacle to the pipes that remain to be routed.

Remark: In the basic routing problem we assume that every pipe connects two terminals. Though this is the most common situation in practice, there may exist some pipes, called *nets*, which have more than two terminals. The channel and path generation algorithms implemented in our pipe router have been extended to handle the case of these multi-terminal nets. The extension works as follows. The algorithm selects two terminals of the net and constructs a channel connecting these terminals, regardless of the other terminals. It then selects another terminal among the remaining ones and searches for a channel connecting this terminal to any of the cells in the channel connecting the first two terminals. This process is repeated for each of the remaining terminals in the net, yielding a multi-branch channel. A path is extracted from this channel by connecting the terminals in the same order as for the construction of the channel. Figure 3.11 shows a route generated for a four-terminal net.

Figure 3.11: An example of a multi-terminal net

3.6 Backtracking

If the search of CG_k fails to produce a channel in the decomposition of the free space in C_k, the algorithm must backtrack. That is, it must change the routes of some of the previously routed pipes to make room for the current pipe $\mathbf{P}_k$. Backtracking requires the pipe router to decide *which* routes to change and *how* to change them. One must also avoid running iteratively through the same cycle of failures.

A simple approach consists of applying a *chronological backtracking strategy* by changing routes one by one in the reverse chronological order of their generation. However, this approach is quite inefficient, because it often leads to changing routes that are not relevant to the failure of finding a route for $\mathbf{P}_k$. This drawback is illustrated by the example shown in Figure 3.12a. In this example, $\mathbf{P}_1$ and $\mathbf{P}_2$ have been routed in sequence. Then the algorithm fails to find a route for $\mathbf{P}_3$. A chronological backtracking strategy would first change $\mathbf{P}_2$, while it is obvious that the current route of $\mathbf{P}_1$ is the only one that is responsible for the failure.
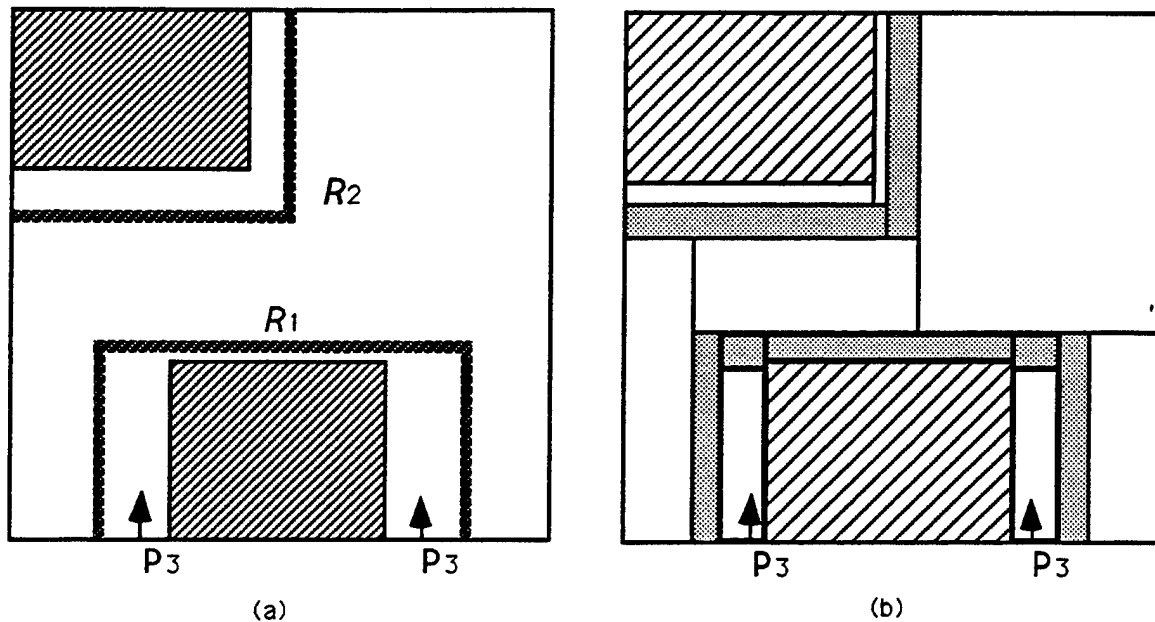
Figure 3.12: Identification of relevant routes to rip up

Inspired by dependency-directed backtracking techniques [Latombe, 1976 and 1979] [Stallman and Sussman, 1977], we have developed a more sophisticated backtracking strategy. If the search of CG_k fails to find a route for $\mathbf{P}_k$, we augment the connectivity graph CG_k by adding those rectangloid cells which are occupied by the bounding approximations of the previously generated routes. (As we will see in a moment, some routes may be "protected"; only those cells corresponding to the other, unprotected routes are included in the augmented graph.) We search the augmented connectivity graph CG'_k for a channel. The cells of the generated channel that were not in the previous graph, if any, allow the router to identify which routes are relevant to the failure. (If no channel can be built in CG'_k, we know that at the resolution of the approximation of the free space there is no route for $\mathbf{P}_k$ regardless of the other routes, or that a mistake has been made in backtracking.) The number of planned routes to be changed in order to make room for $\mathbf{P}_k$ depends on the channel generated by the search of CG'_k (assuming that one can be generated). We can reduce this number by searching CG'_k with an A* algorithm using an appropriate evaluation function. We then only change those routes that occupy cells in the channel generated by the

search of CG'_k. Figure 3.12b illustrates the application of this backtracking approach. Both the empty cells (shown in white) and the cells occupied by the routes $\mathcal{R}_1$ and $\mathcal{R}_2$ (shown in grey) are in CG'_3. The channel whose cells are shown with bold contours is produced by the search of CG'_3. The grey cells included in this channel are traversed by the route $\mathcal{R}_1$ only. The route $\mathcal{R}_2$ has no intersection with the channel and hence need not be changed.

Once the algorithm has decided which routes to change, it still has to decide how to change them. One approach would consist of locally modifying routes to free all the cells that are included in the channel generated by searching CG'_k. This approach could work well if there were available empty space around these routes for. Otherwise the local modification of a route would require other routes to be changed and controlling the propagation of changes could be difficult to manage. Therefore, in our pipe router, we adopted a more radical, but incomplete, approach which consists of ripping off all the routes that should be changed and re-routing them after a route has been planned for the current pipe $\mathbf{P}_k$. This essentially corresponds to routing some of the pipes in a different order.

Ripping off routes increases the free space in $\mathcal{C}_k$. Rather than recomputing a new decomposition of the free space, however, we consider the cells forming the bounding approximation of the C-obstacles $C\mathcal{R}_j$ corresponding to the ripped off routes and we include them in the cell decomposition of the new free space together with the cells that were previously computed with all the routes $\mathcal{R}_1$ through $\mathcal{R}_{k-1}$ being present. (Actually, this step is slightly more complicated and takes into account the fact that each of these cells may have a non-zero intersection with cells approximating the C-obstacles $C\mathcal{B}_i$ and the C-obstacles $C\mathcal{R}_j$ corresponding to remaining pipe routes.) To reduce free space fragmentation (and hence ultimately reduce the cost of the search), we apply simple heuristics to merge adjacent cells together into larger ones. The new decomposition yields a new graph CG_k which contains the channel found for $\mathbf{P}_k$ in CG'_k.

In order to make the approach work properly, however, several issues still have to be addressed. First, after we generated $\mathcal{R}_k$, we need to re-plan the ripped off routes.
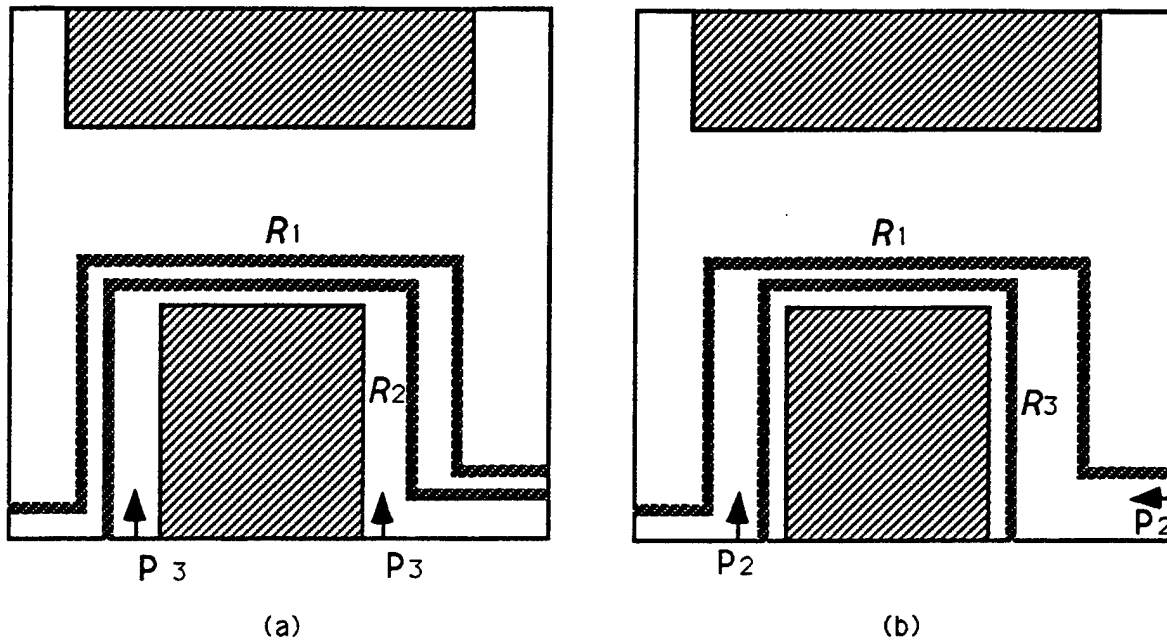
Figure 3.13: Interaction among routes

Re-planning these routes may require other routes to be retracted as illustrated in Figure 3.13. In this example, P_1 and P_2 have been routed. Assume that $\mathcal{R}_2$ is then ripped off in order to make room for the pipe P_3 and that the route $\mathcal{R}_3$ is generated as shown in Figure 3.13b. Planning a new route for P_2 then causes either $\mathcal{R}_1$ or $\mathcal{R}_3$ to be removed, which may cause infinite looping if $\mathcal{R}_3$ is chosen rather than $\mathcal{R}_1$. To avoid infinite looping, we make use of the **net-protection** mechanism described below.

When the route of a pipe P_j is ripped off to make space for routing P_k, P_j remembers P_k in a *protection list* PL_j. When a new route $\mathcal{R}_j$ is generated for P_j, the current routes of the pipes in PL_j are *protected* against P_j. This means that if the generation of $\mathcal{R}_j$ fails, they should not be ripped off to make room for P_j. If the algorithm ultimately succeeds in re-routing all the pipes P_j ($j < k$) whose routes were ripped off, P_k is removed from the corresponding lists PL_j. To illustrate how protection works, consider the previous example (Figure 3.13). When we remove $\mathcal{R}_2$ to route P_3, we store P_3 in PL_2. When we re-route P_2, we must remove either $\mathcal{R}_1$

or $\mathcal{R}_3$. Since $\mathcal{R}_3$ is protected, we decide to rip $\mathcal{R}_1$ off. This allows us to plan a new route $\mathcal{R}_2$, and next a new route $\mathcal{R}_1$. $\mathbf{P}_3$ is then removed from PL_2.

Figure 3.14: A more difficult example

However, the channel found by searching CG'_k may not be unique. If this is the case, different subsets of routes could have been ripped off in order to make room for $\mathbf{P}_k$, and perhaps the algorithm chose the wrong subset. This possibility requires the above protection mechanism to be completed. As an illustration, consider Figure 3.14. Suppose that having failed to plan a route for $\mathbf{P}_4$ (Figure 3.14a), the algorithm decides to rip $\mathcal{R}_3$ off (hence, $\mathbf{P}_4$ is stored in PL_3). After $\mathbf{P}_4$ has been routed as shown in Figure 3.14b, the algorithm tries to re-route $\mathbf{P}_3$ and it fails. It then identifies the route of $\mathbf{P}_4$ as being responsible for this failure. Since $\mathbf{P}_4$ is protected, its only remaining alternative is to withdraw the previous decision to rip $\mathcal{R}_3$ off for routing $\mathbf{P}_4$. That is, it re-establishes the situation shown in Figure 3.14a and attempts to route $\mathbf{P}_4$. But, this time, it rips $\mathcal{R}_1$ off, rather than $\mathcal{R}_3$.

The above bactracking algorithm allows the pipe router to eventually consider all the possible orderings on the pipes, without looping. It has been implemented and

works well for most reasonable routing problems.

This backtracking algorithm is also directly applicable to a three-dimensional workspace. The main difference in the three-dimensional case is that mis-routing a pipe is more likely to cause unnecessary turns in some of the routes planned later rather than search failures. For example, in the example of Figure 3.12a, if the workspace was three-dimensional, P_3 could be routed, but it would have at least two extra turns. In our pipe router, we use a heuristic function to activate the backtracking algorithm when it detects pipes with an unrealistic number of turns.

3.7 Implementation and Experimentation

The basic sequential pipe router described above has been implemented in Allegro Common Lisp running on a Macintosh II for a two-dimensional and a three-dimensional workspace. In our current implementation, a three-dimensional workspace is discretized into multiple two-dimensional layers. Figure 3.15 shows an example where two pipes are routed one after the other in a three-dimensional workspace with two layers. The window "Workspace" shows the projection of the pipe routes into the xy-plane of the workspace. The first pipe lies entirely in the first layer; the second pipe lies partly in the first layer and partly in the second layer (darkest portion). The windows "Layer 1" and "Layer 2" show the first and the second layers of the corresponding configuration space. Figure 3.16 continues the previous example with additional pipes routed in the same workspace.

We have experimented with many additional routing problems, some of which are shown in Figures 3.17 and 3.18. In these examples, only the "Workspace" window is shown. Based on our experiment, the time it takes to route each pipe increases as more pipes are routed in the same workspace. Furthermore, the pipes that are routed later are more likely to have more turns and a suboptimal length. Both of these problems are due to the fact that the space is getting more cluttered as more pipes are routed. We also discovered that the ordering of the pipes affects both the time it takes to route the pipes and how good the routes are.
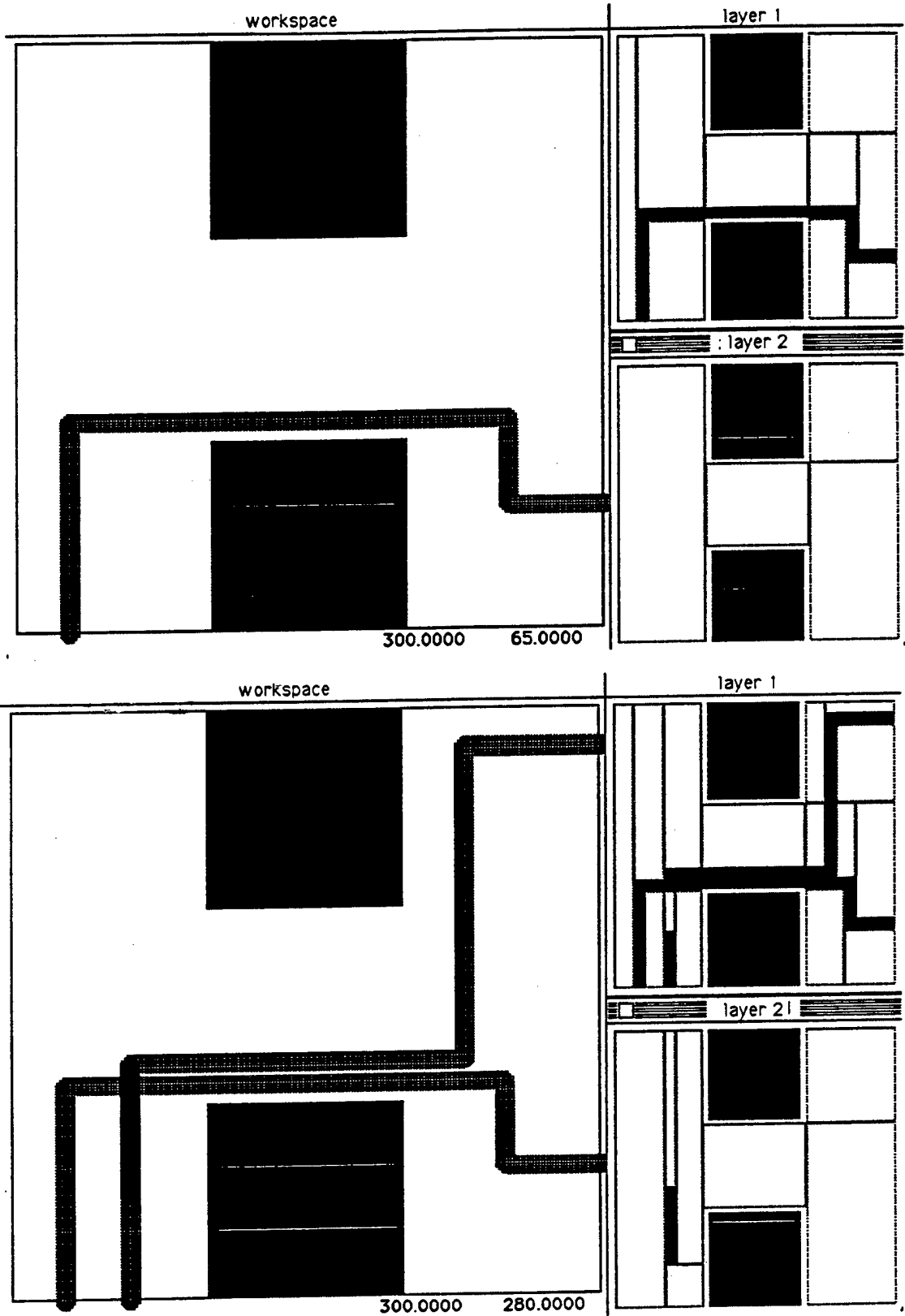
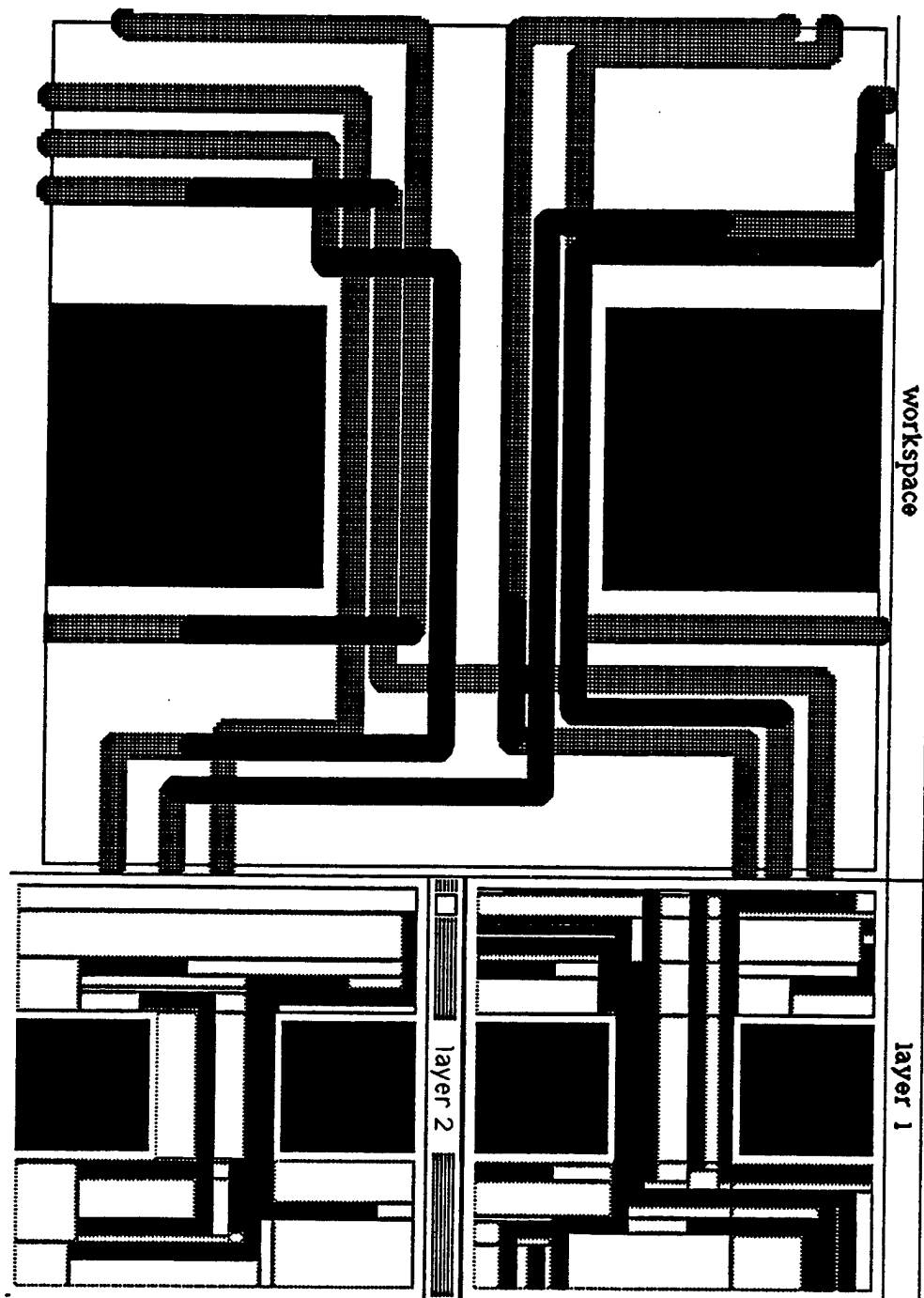Figure 3.15: An example from the basic sequential pipe router

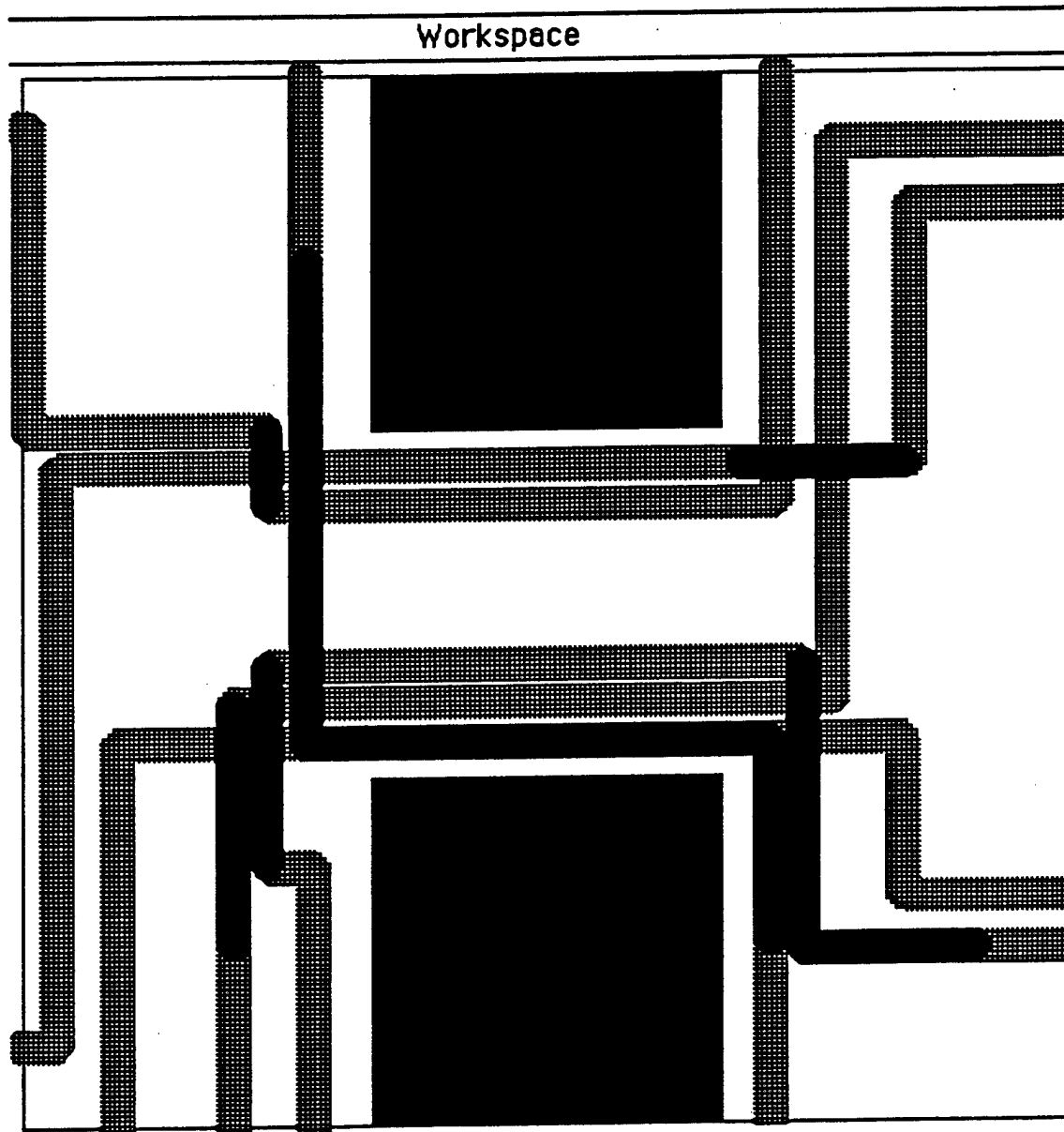Figure 3.16: Additional pipes routed in the same workspace as the previous example

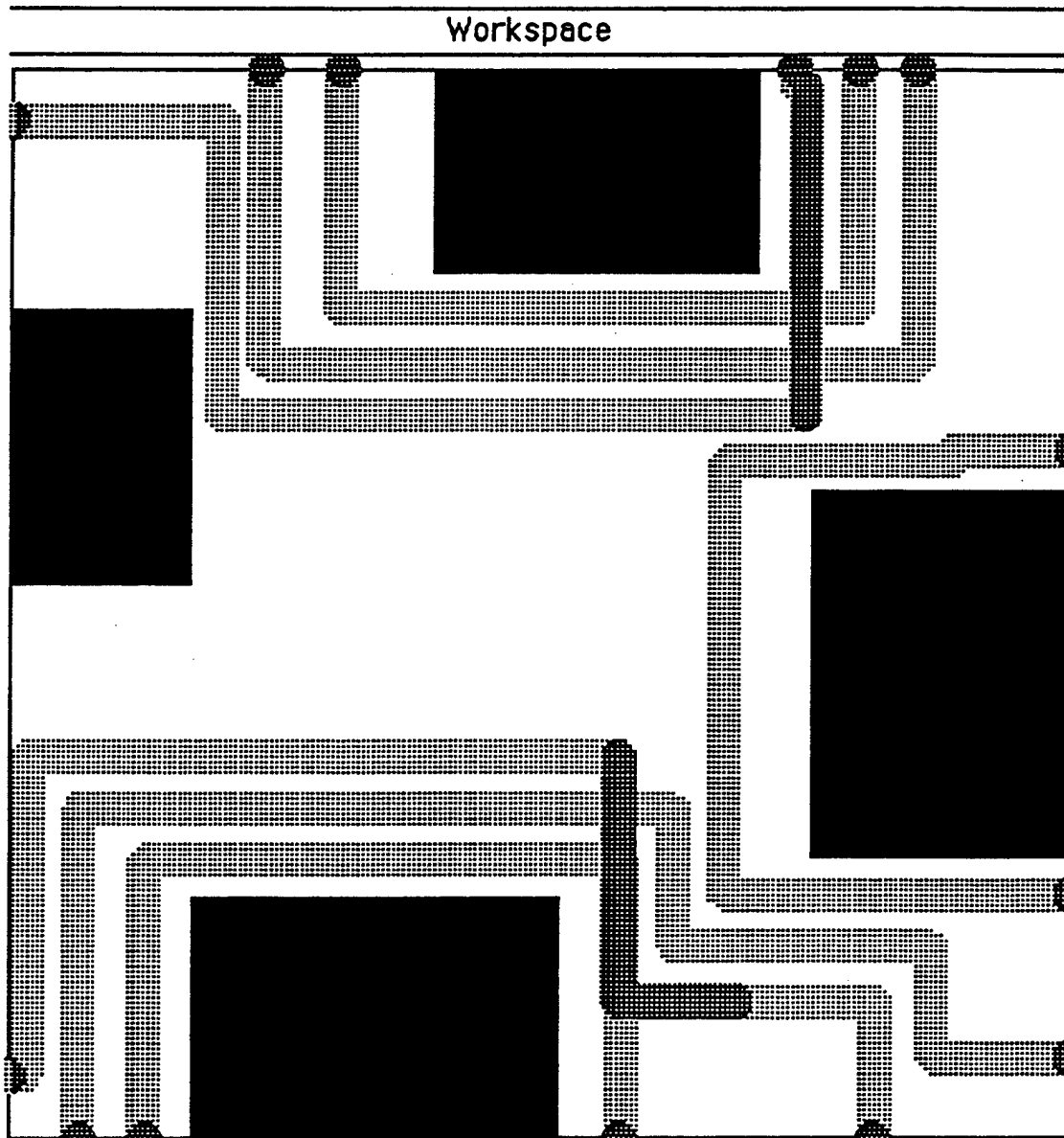Figure 3.17: Another example from the basic sequential pipe router

Figure 3.18: Another example from the basic sequential pipe router

3.8 Limitation

The bactracking algorithm described above is not complete, and there are two reasons. First, the above protection mechanism is too strict, as illustrated in the example of Figure 3.19. Routing P_2, after P_1 has been routed as shown in Figure 3.19a, causes $\mathcal{R}_1$ to be ripped off and therefore causes P_2 to be remembered in PL_1. When the algorithm attempts to re-route P_1 after it has routed P_2, as shown in Figure 3.19b, it fails. Recovering from the failure would require to rip the current route of P_2 off, but it is protected. On the other hand, the algorithm cannot withdraw its decision to rip the route of P_1 off since there was no other possible responsible pipe for the failure to route P_2. Nevertheless, there exists a solution for the routing problem shown in Figure 3.19c.
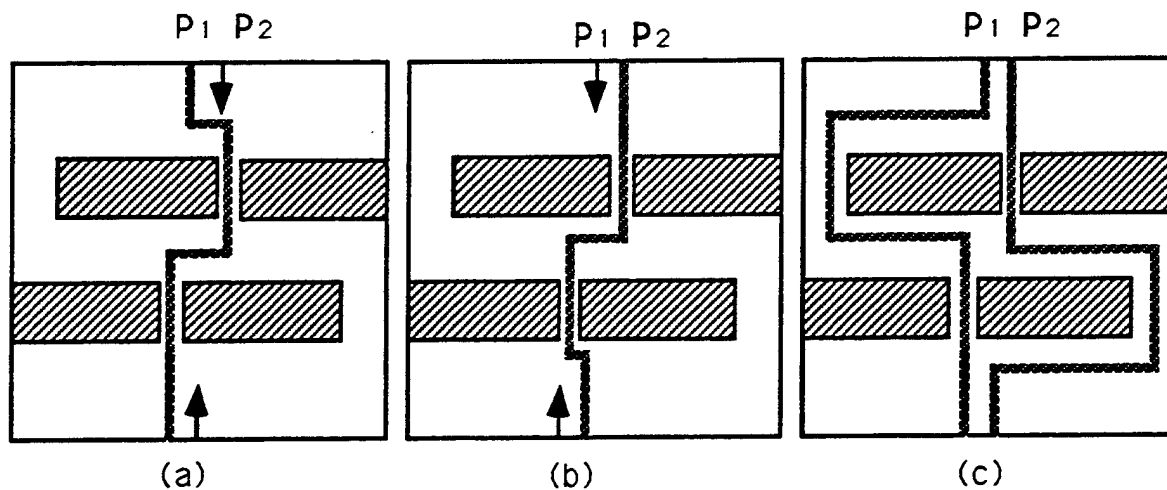
Figure 3.19: An example where the protection mechanism fails

The basic reason why the backtracking algorithm fails in this example is that the protection mechanism identifies pipes as being responsible for the failures, while pipes routed in a certain way are the actual causes for the failure. In Figure 3.19a, the failure to find a route for P_2 is caused by P_1 with its current route. In order to be more complete, the backtracking algorithm should be able to identify a segment of the pipe route of P_1 as being responsible for the failure, and then only changes

that segment of the route. Modifying the algorithm accordingly would not be very difficult, but it would result in a much larger solution space to explore, and therefore a much less efficient pipe router. In fact, the backtracking algorithm implemented in our system is even more radical. In the example of Figure 3.14, after it has re-established the situation shown in Figure 3.14a and before it attempts to route P_4 again, it stores P_3 in a *permanent* protection list attached to P_4. This list prevents the routing of P_4 to ever cause the removal of the route of P_3 in the rest of the routing process. Permanent protection slightly worsens the incompleteness of the pipe router, but it further increases its efficiency.

The other cause of incompleteness is that the algorithm does not consider the possibility of modifying the path of a route *within* a channel. This second cause is partially eliminated in the *parallel routing* approach described in the next chapter.

3.9 Summary

In this chapter we started to explore the problem decomposition approach in the context of pipe routing. In particular,

- we have presented a spectrum of approaches to the pipe routing problem which can be viewed as a special case of the multi-robot path planning problem;

- we have developed a sequential pipe routing algorithm based on the approximate cell decomposition method;

- we have developed a selective backtracking algorithm with a net-protection scheme which prevents the algorithm from looping and allows it to detect and recover from backtracking error;

- we have implemented a sequential pipe router based on the techniques developed here and experimented with this router with many pipe routing examples.

Chapter 4

Parallel Pipe Routing

4.1 Motivation and Approach

The problem decomposition approach transforms the problem of routing many pipes into a set of subproblems, each consisting of finding a route for one pipe. Since no two routes should overlap, the subproblems interact. If we solve these subproblems sequentially, as discussed in the previous chapter, the routing of one pipe results in additional constraints that apply to the pipes that have not been routed yet. Backtracking is required when no route can be found for a pipe, i.e. the subproblem of routing that pipe is over-constrained. When backtracking occurs too frequently due to strong subproblem interaction, the sequential routing approach becomes inefficient.

In this chapter our goal is to reduce backtracking by delaying the choice of a specific solution (i.e. a specific route) to a subproblem until the interactive constraints among the subproblems have been made sufficiently explicit. This idea has been previously investigated in AI where it is known as the *least-commitment* approach [Mackworth, 1977]. One early application of this idea was to the labeling problem in computer vision ([Huffman, 1971], [Clowes, 1971], and [Waltz, 1975]). In a blocks world image, the labeling of each visible block vertex is treated as a subproblem. Each vertex has a small number of possible labels. The label of a vertex constrains and is constrained by the labels of the adjacent vertices. The Waltz algorithm [Waltz, 1975] for vertex labeling first tries to reduce the set of possible labels for a vertex by

checking for consistency between adjacent vertices before a label is selected for it. A label is removed from the set of possible labels for a vertex iff it is inconsistent with every label in the label set of an adjacent vertex. Other forms of least-commitment ideas have also been explored, in particular, in various AI planning systems (e.g. DEVISER [Vere, 1985], MOLGEN [Stefik, 1981], NOAH [Sacerdoti, 1975], NONLIN [Tate, 1976], and SIPE [Wilkins, 1984]), in scheduling problems [Allen, 1983] [Rit, 1986], and other constraint satisfaction problems [Mackworth and Freuder, 1984].

The pure least-commitment approach is inappropriate for pipe routing problems, unless the space is extremely cluttered. Indeed, by considering all potential interactions among pipes, it defies the purpose of the problem decomposition approach, which is to achieve efficiency by ignoring unimportant interactions among pipes. In fact the pure least-commitment approach to pipe routing is strictly equivalent to the centralized approach discussed in Subsection 3.2.2, in which planning is carried out in the composite configuration space which is equal to the Cartesian product of the configuration spaces of the various pipes. Hence, its computational complexity is exponential in the number of pipes.

Consider the spectrum of approaches to multi-robot path planning discussed in Section 3.2. As we move from the centralized approach to the sequential approach with backtracking, we perform more and more searching, but less and less geometric reasoning. The sequential approach is mostly search. The search takes place at two levels: 1) for finding an ordering of the pipe to route and 2) for finding a channel for each pipe. Geometry plays a limited role: in decomposing the space and in checking for interference among pipes. At the opposite end of the spectrum, the centralized approach is mostly geometric reasoning and the search effort is minimal. The sequential approach is not efficient when the interactions among the pipes are strong enough to cause many backtracking operations. The centralized approach would not be appropriate either, unless the interactions among the pipes are so strong that they warrant complete consideration of their interactions.

In this chapter we propose and investigate an approach for dealing with stronger pipe interactions. It lies between the centralized approach and the sequential approach

with selective backtracking. We call it the *parallel pipe routing* approach. In the sequential pipe routing approach, a pipe route is generated in two steps: first a channel is generated and then a route is chosen within the channel. These two steps are performed for all the pipes sequentially. In the parallel pipe routing approach, we first generate channels for all the pipes without computing a route for any of them. If these channels do not overlap, then a route can be generated for each pipe within its channel, independently of the other pipes. But if some channels overlap, the interactions among the corresponding pipes are considered before specific routes are generated for these pipes. For each pipe, this is accomplished by removing from its channel the regions that cause harmful interactions with other pipes.

We first describe the parallel routing algorithm when the workspace is two-dimensional. Then we present its extension to the three-dimensional case.

4.2 Two-Dimensional Case

We illustrate the parallel routing algorithm in a two-dimensional workspace with the simple example shown in Figure 4.1. In this example there are three obstacles, B_1, B_2, and B_3, and three pipes to be routed, N_1, N_2, and N_3, which are specified by (T_1^1, T_1^2, r), (T_2^1, T_2^2, r), and (T_3^1, T_3^2, r), respectively. Throughout this chapter, for simplification, we assume that all the pipes have the same radius. The removal of this assumption is discussed in Section 4.5.

4.2.1 Channel Generation

The first step of the parallel routing algorithm is to generate a channel for each pipe. As with the sequential routing algorithm, we first decompose the configuration space into rectangular cells. The EMPTY cells are used to build the connectivity graph CG. In our pipe routing example, a possible decomposition of the space is shown in Figure 4.2. The EMPTY cells are denoted by $C_1, \ldots, C_8$. Lower case letters are used to denote the end points of the edges between adjacent cells. For example, ab is the edge between cells C_2 and C_3. Since all pipes have the same radius, the same decomposition
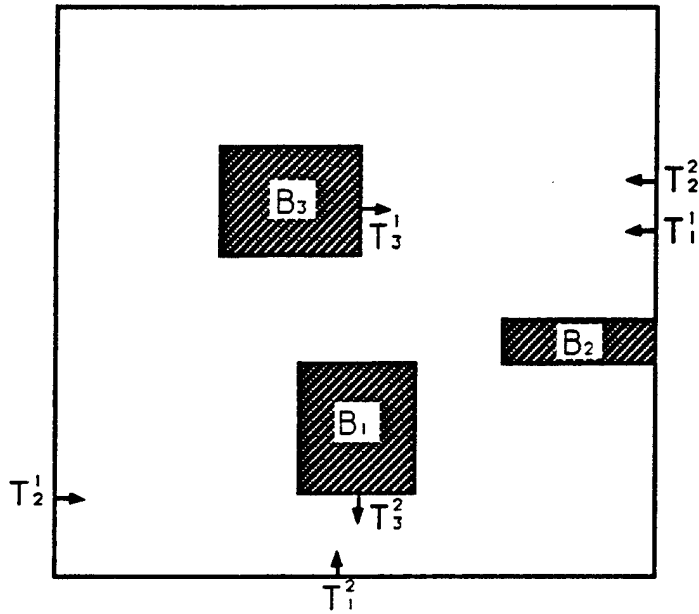
Figure 4.1: A two-dimensional pipe routing problem

is used for all the pipes. For each pipe, the graph CG is searched for a channel, i.e. a sequence of cells connecting the two terminals of the pipe. Continuing with our example, let us assume that the algorithm finds the channels $\{C_4, C_1, C_2, C_3, C_8\}$ for N_1, $\{C_4, C_1, C_7, C_6, C_2, C_5\}$ for N_2, and $\{C_1, C_2, C_5, C_8\}$ for N_3. These three channels are shown in grey in Figure 4.3. The keen reader may observe that the channels have been "well" chosen, so that routes actually exist for the three pipes within these channels. In particular, while there exists a shorter channel $\{C_4, C_1, C_2, C_5\}$ for N_2, this channel and the other two channels together would not contain solution routes for the three pipes. In Subsection 4.2.6, we will discuss the issue of changing a channel when it is detected that an erroneous choice has been made.

When each channel is generated, the *pipe set* of each of the cells traversed by the pipe is updated. For example, when the channel $\{C_4, C_1, C_2, C_3, C_8\}$ for N_1 is generated, the triplet (N_1, de, cd) is stored in the pipe set of cell C_1 to remember that N_1 goes through this cell from edge de to edge cd. When all three channels are constructed, the pipe set attached to C_1 is $\{(N_1, de, cd), (N_2, de, cd), (N_3, T_3^1, cd)\}$ where the terminal T_3^1 stands for a zero-length edge segment.
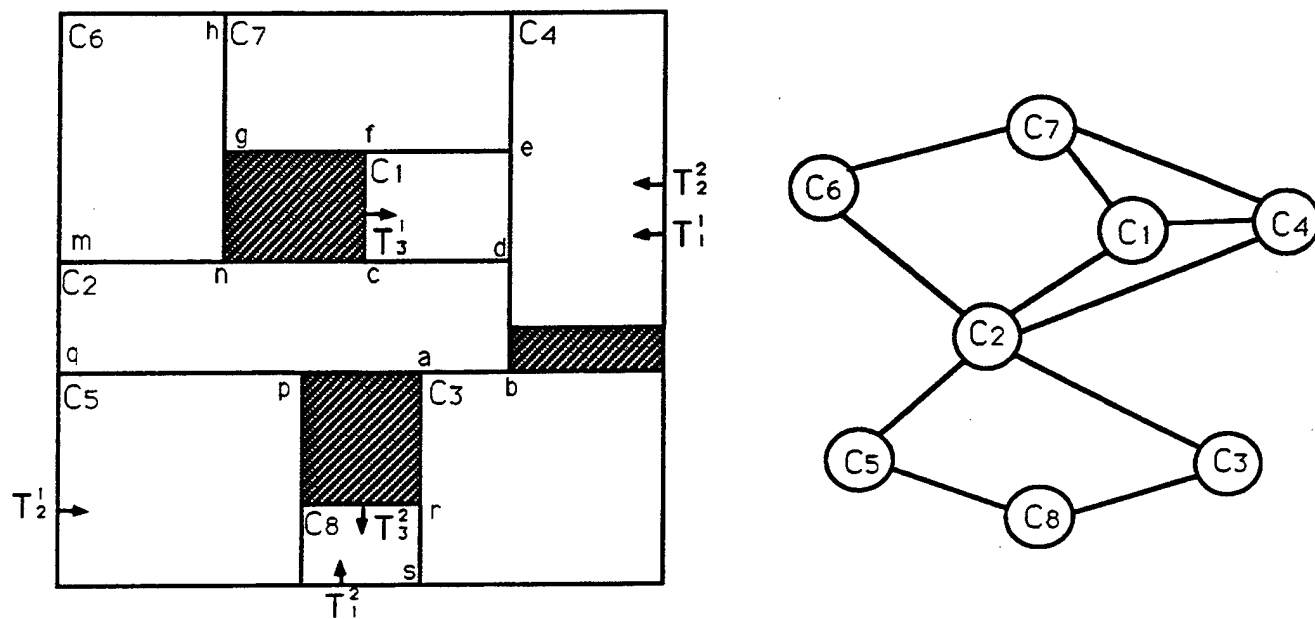
Figure 4.2: Cell decomposition and associated connectivity graph
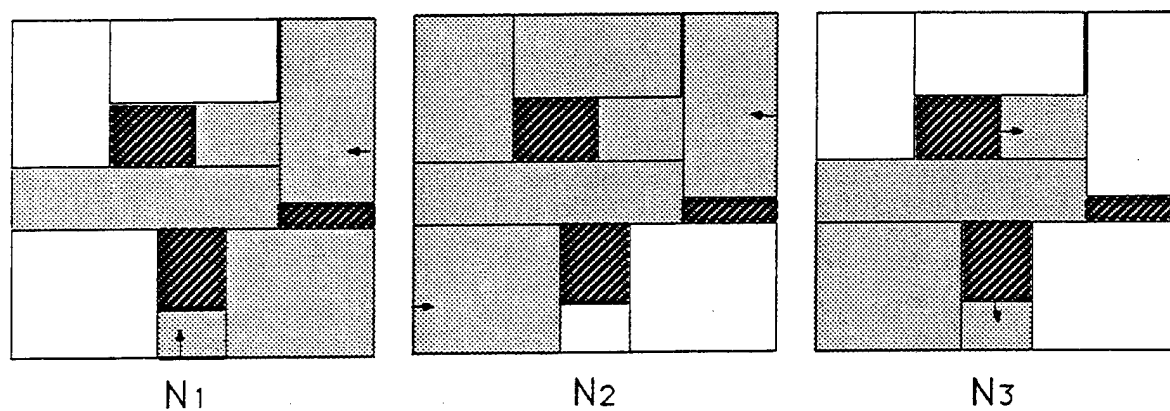
Figure 4.3: Channels

4.2.2 Considering Pipe Interactions within Channels

Once all the channels have been generated, the remaining problem is to find a specific route within each one. Usually we cannot arbitrarily select a route within each channel because channels may overlap. When two channels overlap, the two corresponding pipes are said to "interact" in the overlapping cell(s). For example, all three channels shown in Figure 4.3 share cell C_1. Hence, each pipe must share this cell space with the other two. To make sure that the route selected for N_1 will not later prevent the other two pipes to be routed through C_1, we restrict the region within C_1 that N_1's route is allowed to traverse. This is done by removing the minimal area needed by N_2 and N_3 to traverse C_1. In this way, we refine the channel of N_1 before a specific route is generated for N_1. By doing so, we anticipate and eliminate some harmful interaction among pipes. The channels of the other two pipes are refined in the same manner.

Channel refinement takes place in every cell that is traversed by more than one pipe. In fact, once channel generation is completed, we regard the pipe routing problem as a collection of subproblems. Each subproblem is specified by a cell C_k and its associated pipe set $\{(N_{k_1}, edge1_{k_1}, edge2_{k_1}), \ldots, (N_{k_m}, edge1_{k_m}, edge2_{k_m})\}$. It requires finding a route connecting a point in $edge1_i$ to a point in $edge2_i$ for each pipe N_i, $i \in \{k_1, k_2, \ldots, k_m\}$, in the pipe set of C_k. We call the portion of a pipe N_i within a cell C_k a *pipe segment* and we denote it by $n_i^{C_k}$ (the superscript will be omitted whenever it is understood from the context). Figure 4.4 illustrates this subproblem viewpoint to our pipe routing example. For instance, the subproblem associated with cell C_1 is to find the routes of three pipe segments n_1, n_2, and n_3, connecting a point in *de* to a point in *cd*, a point in *de* to a point in *ef*, and a point in *cd* to T_3^1, respectively. Since a route must be continuous between two adjacent cells, the various subproblems may not be completely independent.

In order to refine the portion of the channel corresponding to a pipe segment n in a cell C, we must first characterize the position of n relative to the other pipe segments traversing C. This relative position is often, but not always, determined
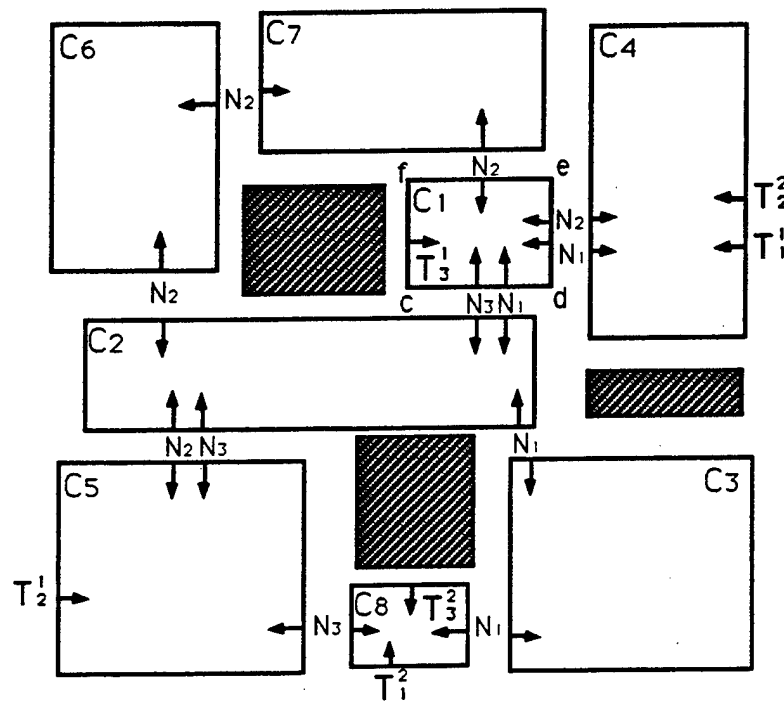
Figure 4.4: Subproblems in parallel pipe routing

by the relative locations of the edges where the pipes enter and exit[1] the cell. For example, in Figure 4.5a, there are two pipe segments n_1 and n_2 traversing the cell C. The segment n_1 enters through the left edge, while the segment n_2 enters through the right edge. Both pipe segments exit through the bottom edge. Since the two segments must not intersect, we can derive that n_1 lies "on the left of" n_2, a spatial relation that will be formally defined below. Therefore, we should restrict the region for n_1 to the left side of the cell (the grey region in Figure 4.5a) leaving just enough room on the right for n_2. Similarly we should restrict the region for n_2 to the right side of the cell.

However, inferring the relative position of two pipe segments just by looking at where they enter and exit the cell is not always possible. For instance, if two pipes enter the cell through the same edge and exit it through the same edge, as is the

---

[1]Since pipes are not oriented, the two notions of entrance and exit of a pipe are completely interchangable.

case for the two segments in the cell C_1 shown in Figure 4.5b, we cannot determine which one of the two segments is "on the left of" the other. In this example, we must look at the neighboring cell C_2 to determine the actual relative position of the two segments in C_1. The relative position of the two pipe segments in C_2 implies that n_1 enters C_1 "on the left of" of n_2, hence determining the relative position of $n_1^{C_1}$ and $n_2^{C_1}$. It is not difficult to imagine situations where one must look several cells away in order to determine the relative position of two pipe segments in a cell.

(a)   (b)
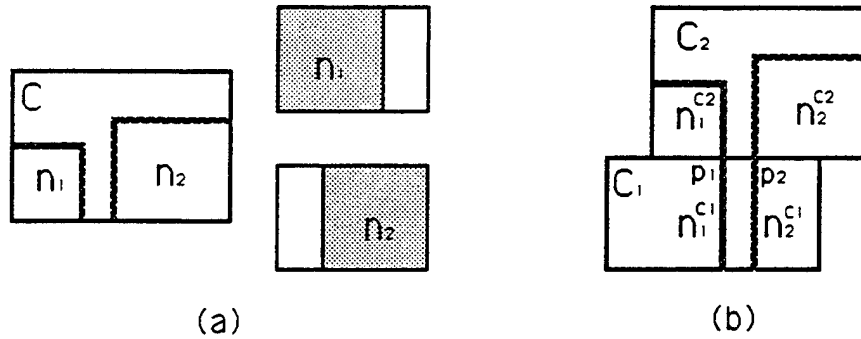
Figure 4.5: Examples of channel refinement

Hence, the interactions among pipes within their chosen channels are considered in two steps: *constraint derivation* and *channel refinement*. We describe these two steps in more detail below.

4.2.3   Constraint Derivation

We describe the relative position of two pipe segments n_1 and n_2 in a cell by means of a *relative position constraint* using two predicates, $<_x$ (which we called "on the left of" in the previous discussion) and $<_y$ ("below"). These predicates are used only for the purpose of channel refinement, i.e. they only describe the simplest relative position between two pipe segments.

- Let L_y be any line parallel to the x-axis. The constraint $n_1 <_x n_2$ expresses that, for any specific routing of N_1 and N_2, if L_y intersects both n_1 and n_2, then the

x coordinate of every intersection of L_y and n_1 is less than the x coordinate of every intersection of L_y and n_2.

- Let L_x be any line parallel to the y-axis. The constraint $n_1 <_y n_2$ expresses that, for any specific routing of N_1 and N_2, if L_x intersects both n_1 and n_2, then the y coordinate of every intersection of L_x and n_1 is less than the y coordinate of every intersection of L_x and n_2.

An atomic relative position constraint is any constraint of the form $n_1 <_x n_2$ or $n_1 <_y n_2$. A relative position constraint is either an atomic relative position constraint, or the conjunction of two such atomic constraints, one with the predicate $<_x$, the other with the predicate $<_y$.

Figure 4.6 illustrates the relative position constraint language. Figures 4.6a, 4.6b, and 4.6c show three examples where pipe segments n_1 and n_2 satisfy the constraint $n_1 <_x n_2$. Figures 4.6d and 4.6e show two examples of segments n_1 and n_2 satisfying the constraint $n_1 <_x n_2 \wedge n_2 <_y n_1$.
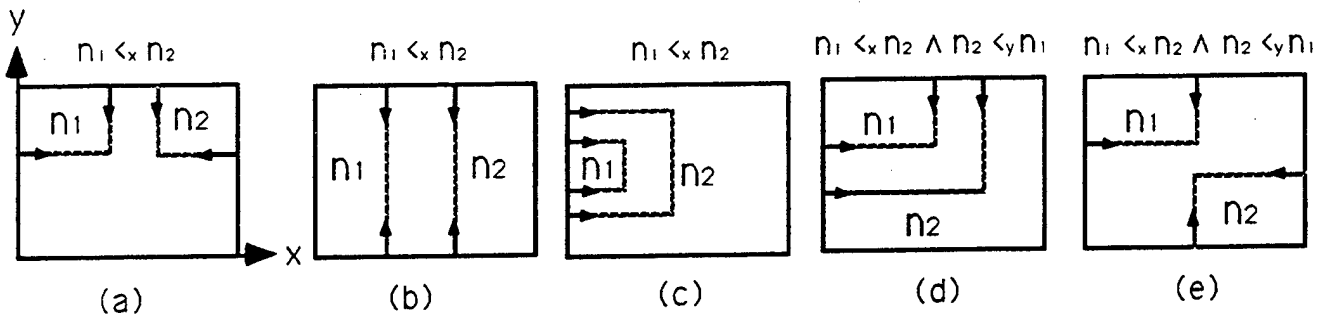
Figure 4.6: Relative position constraints between two pipe segments

As we discussed above, the relative position of two pipe segments in a cell is determined by how they enter and exit the cell, which may itself be determined by the relative position of the two pipe segments corresponding to the same pipes in an adjacent cell. To be more precise, the relative position between two pipe segments depends on the relative position of the extremities of these two segments. A relative position relationship between two segments' extremities is described by a *relative boundary*

constraint. We express these constraints using the same predicate symbols $<_x$ and $<_y$ as above. However, these predicates apply here to extremities of pipe segments in a cell. When two extremities lie in different edges of the cell, the relative boundary constraint between them can be determined from the relative position of these edges. When two extremities lie in the same edge, the relative boundary constraint between them is implied by the relative position constraint of the corresponding two pipe segments in one of the two adjacent cells that shares the edge.

When two extremities, p_1 and p_2, of two pipe segments in a cell C lie in different edges, then the relative boundary constraint between them is determined as follow. Let E_1 and E_2 be the edges in which p_1 and p_2 lie respectively. We have $p_1 <_x p_2$, iff the x coordinate of every point on E_1 is less than or equal to that of every point on E_2, and there exists at least one point on E_1 whose x coordinate is less than the x coordinate of every point on E_2. Similarly we have $p_1 <_y p_2$, iff the y coordinate of every point on E_1 is less than or equal to that of every point on E_2, and there exists at least one point on E_1 whose y coordinate is less than the y coordinate of every point on E_2. This is illustrated in Figure 4.7 where the edges E_1 and E_2 are shown as bold line segments. In Figures 4.7a and 4.7b, we have $p_1 <_x p_2$. In Figures 4.7c and 4.7d, we have $p_1 <_x p_2 \wedge p_2 <_y p_1$.
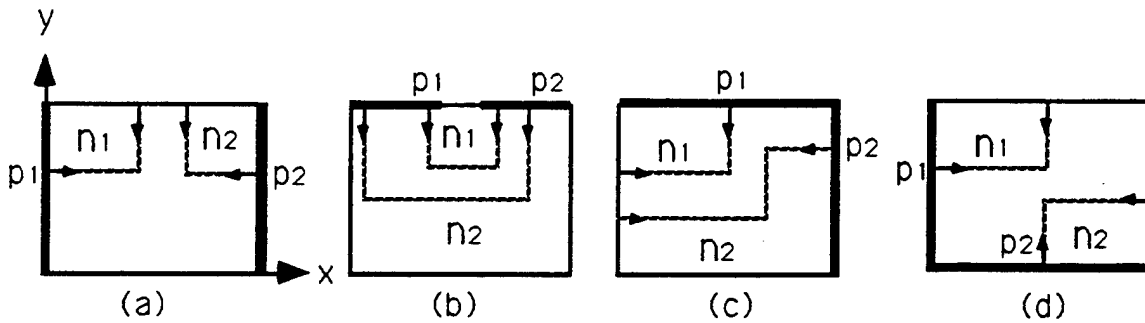
Figure 4.7: Relative boundary constraints derived from the position of the edges

From the relative boundary constraints between the extremities of two pipe segments n_1 and n_2, we can derive relative position constraints between them. Let p_1 and p'_1 be the extremities of n_1 and p_2 and p'_2 be the extremities of n_2. Let us consider

the case where p_1 lies between p_2 and p_2' along the x-axis. There are several cases to consider as shown in Figure 4.8. If p_1 lies in the bottom side (resp. top side) of the cell, we can derive the relative position constraint $n_1 <_y n_2$ (resp. $n_2 <_y n_1$) as shown in Figures 4.8a, 4.8b, and 4.8c. If p_1 lies in the right side or the left side of the cell, then we can derive the relative position constraint $n_1 <_y n_2$ (resp. $n_2 <_y n_1$) iff at least one of n_2's extremities, say p_2, also lies in the same side of the cell as p_1 and we have $p_1 <_y p_2$ (resp. $p_2 <_y p_1$) as shown in Figures 4.8d, 4.8e, and 4.8f. If p_1 lies between p_2 and p_2' along the y-axis, the cases that we need to consider are the same as above except with the x-axis and the y-axis switched. The treatment for the case where p_1' lies between p_2 and p_2' is exactly the same as that discussed above and the treatment for the case where p_2 or p_2' lies between p_1 and p_1' is completely symmetrical.
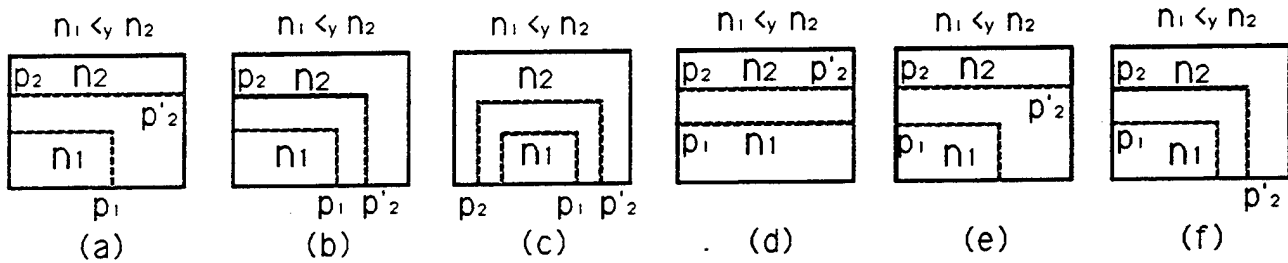
Figure 4.8: Deriving relative position constraints from relative boundary constraints.

When we derive relative position constraints between two pipe segments n_1 and n_2 in a cell C from their boundary constraints, there are three cases to be considered as illustrated in Figures 4.9a, 4.9b, and 4.9c. In these figures p_1 and p_1', and p_2 and p_2' are the extremities of n_1 and n_2, respectively. The relative boundary constraints between extremities that share the same edge are indicated in the figure (note that no relative boundary constraint exists between the extremities in Figure 4.9c). The three cases are:

- Case 1 (Figure 4.9a): Contradicting relative position constraints have been derived from the relative boundary constraints between the extremities of n_1 and n_2. This indicates that the generated channels contain no solution routes for all the pipes.

- Case 2 (Figure 4.9b): A consistent relative position constraint has been derived from the relative boundary constraint(s).

- Case 3 (Figure 4.9c): No relative position constraint can be derived yet from the relative boundary constraint(s).
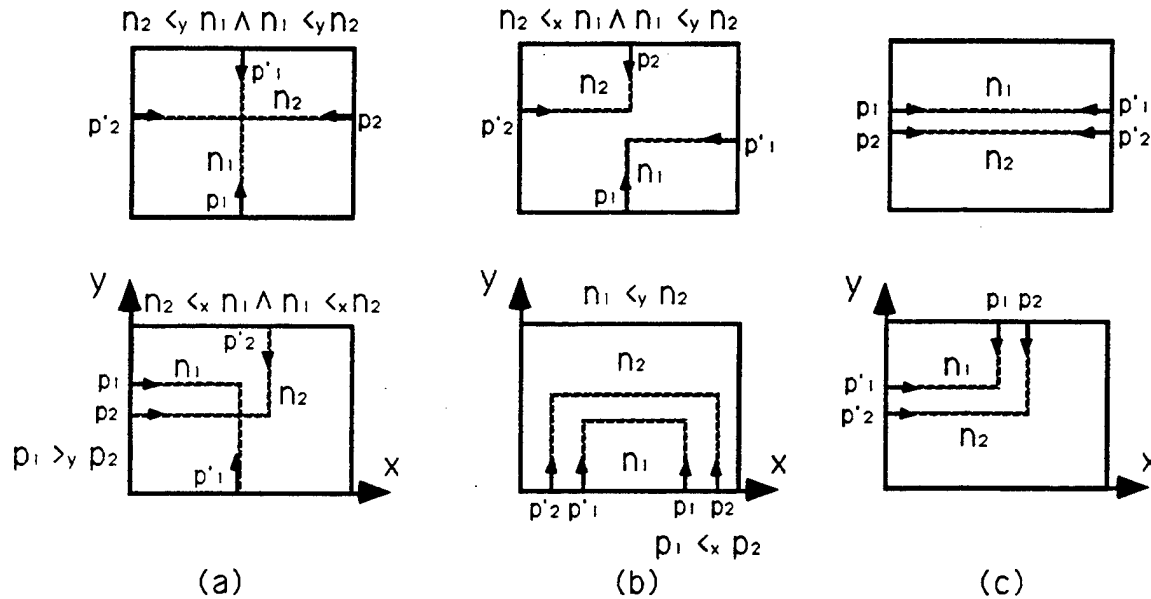

Figure 4.9: Derivation of relative position constraints between two pipe segments.

The derivation of the relative position constraints between pipe segments in the various cells is carried out by a classical constraint propagation algorithm [Waltz, 1975]. Initially, all the cells that are shared by more than one pipe are recorded in a queue Q. Then the algorithm operates iteratively until either a failure has occurred (Case 1 above) or Q is empty. At every iteration, it removes a cell C from Q. For every pipe segment pair n_i and n_j in C, it tries to generate relative position constraints based on the existing boundary constraints on the extremities of these pipe segments. One of the three cases described above occurs. In Case 1, the constraint propagation algorithm terminates with failure. In Case 2, a relative position constraint between the two pipe segments is established and attached to the cell. In addition, a new relative boundary constraint on a pair of extremities of the pipe segments is possibly

derived and, if this is the case, the corresponding adjacent cell is re-entered in Q (if it is not already in Q) for re-examination. In Case 3, nothing is done.

This constraint propagation process is illustrated with our pipe routing example in Figure 4.10:

- Consider cell C_4 first. It is traversed by two segments, n_1 and n_2. Two extremities of these segments are T_1^1 and T_2^2. From their absolute positions we derive that $T_1^1 <_y T_2^2$. The other two extremities, denoted by p_1 and p_2 in the figure, share the same edge between C_4 and C_1. No relative boundary constraint between p_1 and p_2 has been made explicit yet. Since p_1 is between p_2 and T_2^2 along the x-axis, from $T_1^1 <_y T_2^2$, we derive the relative position constraint $n_1 <_y n_2$. In turn, this constraint entails the relative boundary constraint $p_1 <_y p_2$ (Case 2).

- Let us now move to cell C_1. It is traversed by three segments denoted by n_1, n_2, and n_3. Each pair of segments is examined successively:

- For n_1 and n_2, the two extremities p_1 and p_2 share the same edge with a relative boundary constraint $p_1 <_y p_2$ derived in C_4. From this constraint we derive the relative position constraint $n_1 <_y n_2$. This relative position constraint entails no relative boundary constraint for the other extremities of the segments.

- For n_2 and n_3, the entrances and exits of these two segments entail the relative position constraint $n_3 <_x n_2 \wedge n_3 <_y n_2$. No new relative boundary constraint is derived.

- For n_1 and n_3, two extremities, p_1' and p_3', are in the same edge between C_1 and C_2. So far no relative boundary constraint has been derived between them. Using the relative location of the two edges containing the other extremities, we can derive the relative position constraint $n_3 <_x n_1$ which in turn implies the relative boundary constraint $p_3' <_x p_1'$.

The relative positions of the three pipe segments in C_1 are now completely determined by the relative position constraints that have been derived. The propagation algorithm proceeds and considers cells C_2, C_5 and C_8.
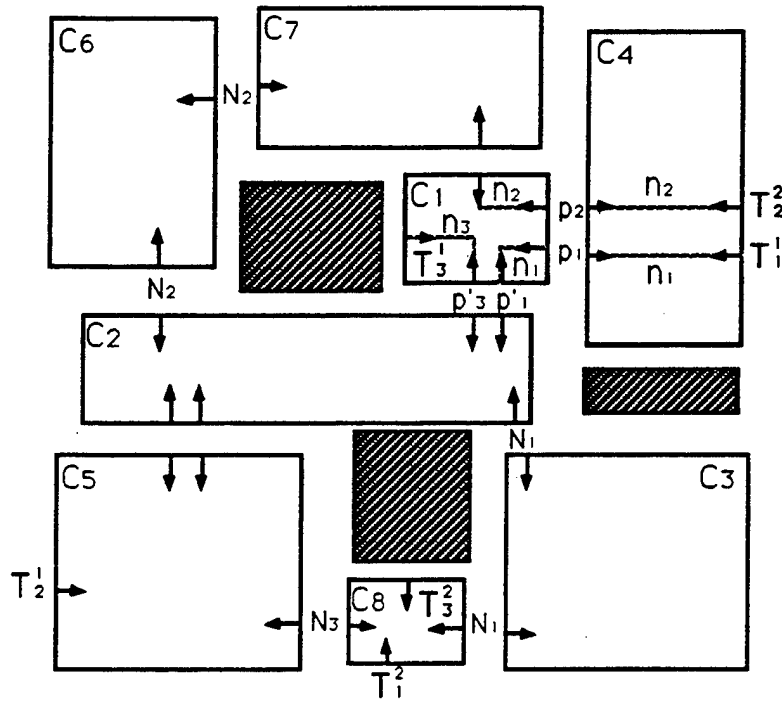
Figure 4.10: An example of constraint derivation

The constraint derivation procedure described above has the following property:

Property 1: If there exists a feasible relative position between every pair of pipe segments, it must be unique. Furthermore, the constraint derivation procedure will terminate in a finite amount of time, returning this unique solution, if it exists, or failure, otherwise.

Proof: We first prove by contradiction the uniqueness of the solution, if it exists. Let us assume that two pipe segments n_1 and n_2 in cell C have more than one feasible relative position between them. It must then be the case that n_1 and n_2 enter C through a common edge and exit C through another common edge. Furthermore, there is no relative boundary constraint between the extremities of these two pipe segments in C. This can be the case iff in each of the two adjacent cells of C, more than one feasible relative position also exists between the two corresponding pipe segments. By applying the same argument to the two adjacent cells of C and the subsequent cells, we can derive the contradiction that the channels for the two pipes

must consist of infinitely many cells. Hence, if a feasible relative position for every pair of pipe segments exists, it is unique.

Let us now prove that the constraint derivation procedure always terminates in a finite amount of time, returning the unique solution, if it exists, or failure, otherwise.

The terminating property comes from the argument that in a finite amount of time, Q becomes empty or a case 1 occurs. In either case, the procedure terminates. Q has a finite number of elements before the procedure starts. New elements are added to Q when a case 2 occurs. Since a case 2 occurs iff a relative position constraint is derived between a pair of pipe segments, and there can be only a finite number of such constraints, there can be only a finite number of elements added to Q. On the other hand, at every iteration of the procedure, an element is removed from Q. Therefore, Q must become empty in a finite amount of time. Of course, it is possible that before Q becomes empty, a case 1 occurs, in which case the procedure also terminates.

It is obvious that if the procedure terminates when a case 1 occurs, it returns failure. We will now prove by contradiction that if the procedure terminates when Q is empty then there is a relative position constraint between every pair of pipe segments. Assume that this is not the case, and pipe segments n_1 and n_2 in C have no relative position constraint between them. Then it must be the case that the element that corresponds to n_1 and n_2 was removed from Q, and has not been put back into Q before the procedure terminates. But this can only be the case if there is also no relative position constraint between the two pipe segments (which correspond to the same pipes as n_1 and n_2, respectively) in each of the two adjacent cells of C. Otherwise when we establish the relative position constraint for the two pipe segments in one of the adjacent cells, the pipe segment pair n_1 and n_2 will be put back into the queue and a relative position constraint will be derived for them because of the relative boundary constraint derived from the adjacent cell. By recursively applying the same argument to the two adjacent cells, we can derive the contradiction that the channels for the two pipes that contain n_1 and n_2 must consist of infinitely many cells. Hence, if the procedure terminates when Q is empty, it has found a relative position constraint between every pair of pipe segments. ∎

4.2.4 Channel Refinement

Assume that the relative position of the pipes within every cell have been determined successfully. The next operation is to refine the channels, one cell at a time. This refinement is done in two steps: *boundary refinement* and *cell refinement*. Boundary refinement consists of refining the edge that a pipe segment is allowed to traverse, using the derived relative boundary constraints between an extremity of this pipe segment and those of the other pipe segments in the same edge. Cell refinement consists of extracting the maximal region that each pipe segment may occupy within a cell without necessarily preventing the routing of the other pipes. It makes use of the derived relative position constraints in that cell.

a. Boundary Refinement. Let us assume that $p_1, \ldots, p_n$ comprise all the extremities in an edge E of the pipe segments in a cell C. Furthermore, let E be described by an interval $[a, b]$ along the x or y axis. The subset of E that p_i is allowed to lie in is also denoted by an interval $[a_i, b_i]$.

The boundary refinement process is carried out in two steps. In the first step, the extremities on E are sorted using the relative boundary constraints on them (at this stage, there necessarily exists a unique relative boundary constraint between every pair of extremities) on E. Let $p_{j_1}, \ldots, p_{j_n}$ be the sorted list. In the second step, for $k = 1, \ldots, n$, the edge interval $[a_{j_k}, b_{j_k}]$ for p_{j_k} is computed by setting

$$a_{j_k} = a + 2r * (k - 1),$$

$$b_{j_k} = b - 2r * (n - k),$$

where r is the pipe radius. This is illustrated in Figure 4.11. Edge capacity overflow occurs if an interval is such that $a_{j_k} > b_{j_k}$.

Property 2: If a pipe is routed through an edge E within the interval allowed by the refined edge interval, it will not prevent the other pipes traversing this same edge E from being routed through it.
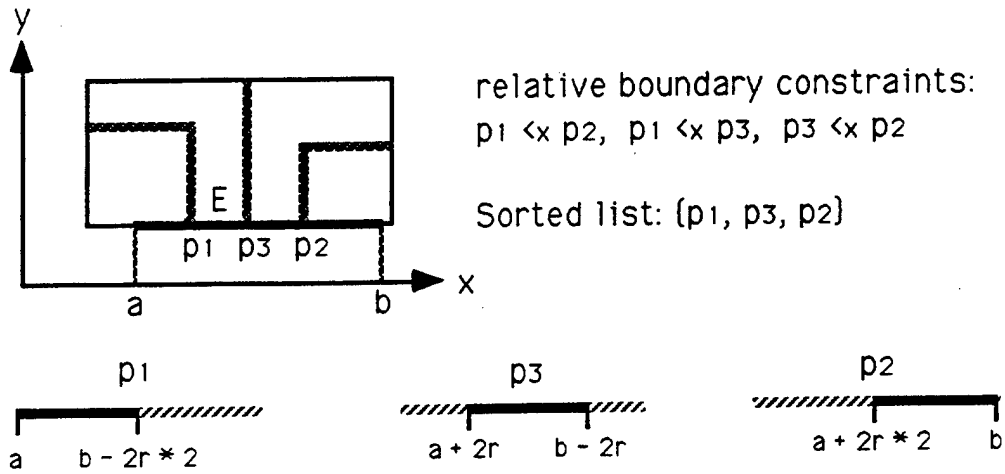
Figure 4.11: Boundary refinement

b. Cell Refinement. Assume that boundary refinement did not detect any edge capacity overflow. Then we proceed to cell refinement. Let us first consider a simple example where a cell C contains two pipe segments n_1 and n_2 with relative position constraint $n_2 <_y n_1$ as shown in Figure 4.12a. We denote the region within C that n_1 (resp. n_2) is allowed to traverse by $FR(n_1)$ (resp. $FR(n_2)$). The goal of cell refinement is to compute $FR(n_1)$ (resp. $FR(n_2)$) such that if a route for N_1 (resp. N_2) traverses C within $FR(n_1)$ (resp. $FR(n_2)$), then it will not prevent the routing of N_2 (resp. N_1) through C. This is accomplished by removing from $FR(n_1)$ the minimum space needed for the routing of n_2 as illustrated in Figure 4.12. Figure 4.12b shows the initial $FR(n_1)$. Figure 4.12c shows the smallest region that n_2 requires to traverse C considering the relative position constraint $n_2 <_y n_1$. We call this region the Minimum Impact Solution of n_2 with respect to n_1 and denote it as $MIS(n_2, n_1)$. Figure 4.12d shows the resulting $FR(n_1)$ after $MIS(n_2, n_1)$ is removed from it. Figures 4.12e, 4.12f, and 4.12g illustrate this same process applied to $FR(n_2)$.

The key step in cell refinement is computing the Minimal Impact Solution, $MIS(n_2, n_1)$. $MIS(n_2, n_1)$ depends on $FR(n_2)$, the relative position constraint between n_1 and n_2, the entrance and exit of n_2, and the geometric characteristics of n_2 (such as its radius and its minimum bend radius). Figure 4.13 illustrates the
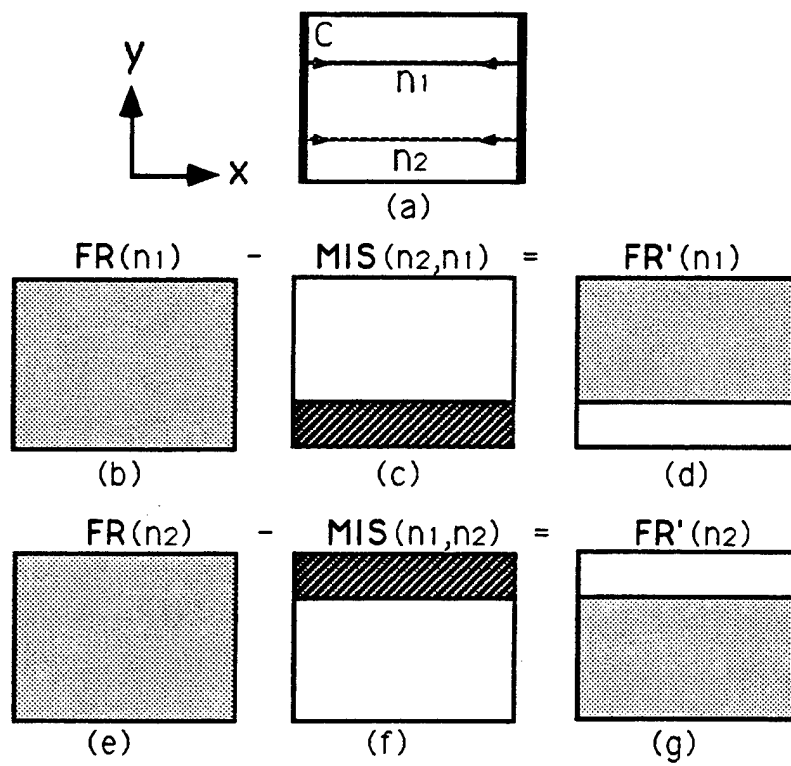
Figure 4.12: An example of cell refinement

construction of $MIS(n_2, n_1)$ in the case when $FR(n_2) = C$. The difference between Figures 4.13a and 4.13b and that between 4.13c and 4.13d illustrates that $MIS(n_2, n_1)$ depends not only on the relative position constraint but also on the entrance and exit of n_2 (The entrance and exit are shown in bold line segments).
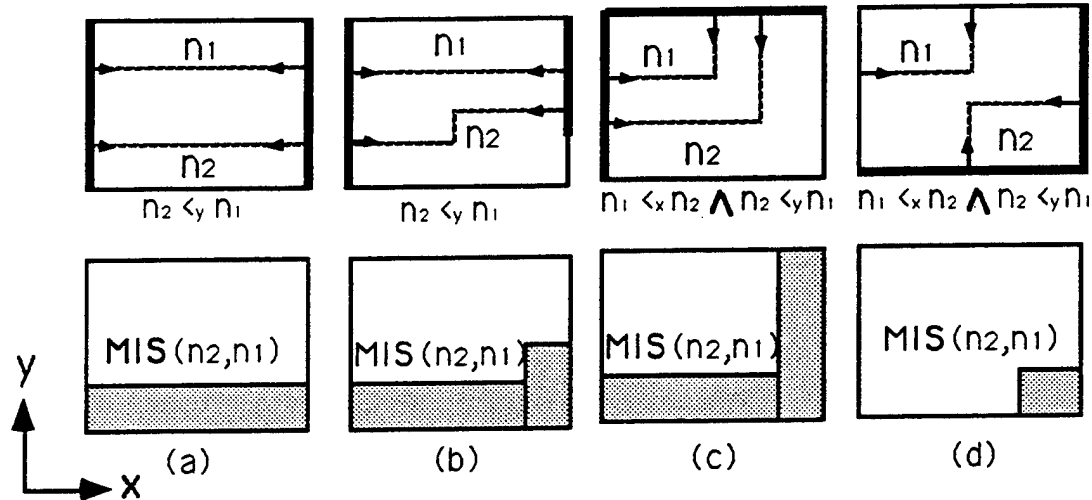

Figure 4.13: Examples of constructing Minimum Impact Solution, the basic cases

If $FR(n_2)$ consists of more than one rectangular region as shown in Figure 4.14, we first search for a "subchannel" within $FR(n_2)$. This subchannel is a sequence of rectangular regions in $FR(n_2)$ that connects the entrance of n_2 to the exit of n_2. For each of the rectangular regions in the subchannel, we can apply the construction process illustrated in Figure 4.13.
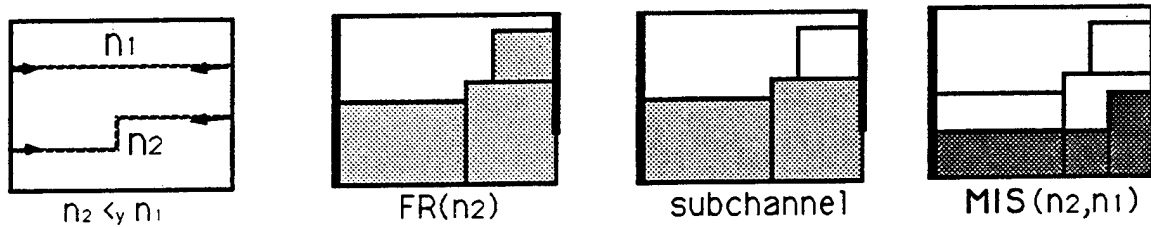
Figure 4.14: An example of constructing MIS: a more complex case

If C contains more than two pipe segments, the cell refinement process is carried out by a constraint propagation algorithm similar to that described above for deriving

relative position constraints. Initially, we create a queue Q that contains all pipe segments, $n_1, \ldots, n_m$, in C. This algorithm operates iteratively until either Q is empty or a failure has been detected, i.e. one or more $FR(n_i)$ has been reduced so much that it no longer connects the entrance to the exit of n_i in C. At every iteration, it removes a pipe segment n from Q. For every pipe segment n_j in Q, it computes the difference $FR(n) - MIS(n_j, n)$. If this difference is a proper subset of $FR(n)$, it sets $FR(n)$ to the difference and puts all pipe segments, except n_j, back to Q if they are not already in it.

We continue to use our routing example to illustrate this cell refinement process. In particular, let us consider cell C_1 as shown in Figure 4.15 (where the feasible region for a pipe segment is shown as a light gray region within which the Minimum Impact Solution is shown in dark gray). Remember that C_1 contains three pipe segments, n_1, n_2, and n_3, with relative position constraints $n_1 <_y n_2$, $n_3 <_y n_2 \wedge n_3 <_x n_2$, and $n_3 <_x n_1$. Initially $Q = (n_1, n_2, n_3)$. In the first iteration, n_1 is removed from Q and $FR_{new}(n_1) = FR(n_1) - MIS(n_2, n_1)$ is computed. Since $FR_{new}(n_1)$ is a proper subset of $FR(n_1)$, $FR(n_1)$ is set to $FR_{new}(n_1)$. Then $FR_{new}(n_1) = FR(n_1) - MIS(n_3, n_1)$ is computed. Again $FR_{new}(n_1)$ is a proper subset of $FR(n_1)$, and $FR(n_1)$ is set to $FR_{new}(n_1)$. Q remains the same, i.e. $Q = (n_2, n_3)$, because both n_2 and n_3 (which are constrained by n_1) are already in Q. Notice that in iterations 4 and 5 no change is made to $FR(n_1)$ and $FR(n_2)$. After iteration 5, Q is empty and hence the cell refinement process for C_1 terminates. The refined feasible regions have the following property:

Property 3: If a route for a pipe segment is chosen within its refined feasible region in a cell C, it will not prevent other pipe segments sharing C from being routed in C, if there exists a set of routes for all the pipes in C.

We apply this algorithm to all the other cells traversed by more than one pipe. The resulting refined channels are shown in Figure 4.16. Property 3 can be generalized to the whole channel; stating that if a route for a pipe is chosen within its refined channel, it will not prevent other pipes from being routed (even though their channels may overlap), if there exists a set of routes for all the pipes.
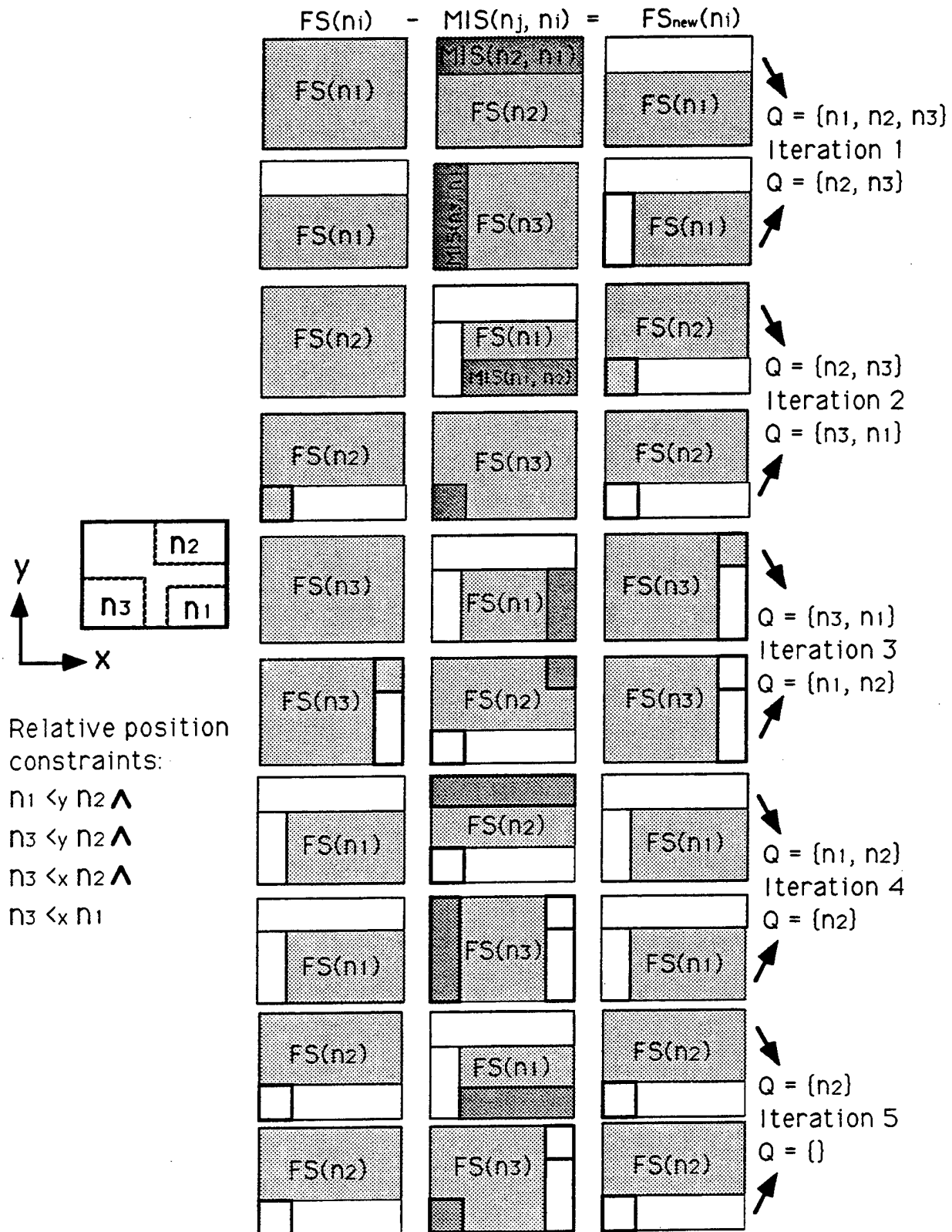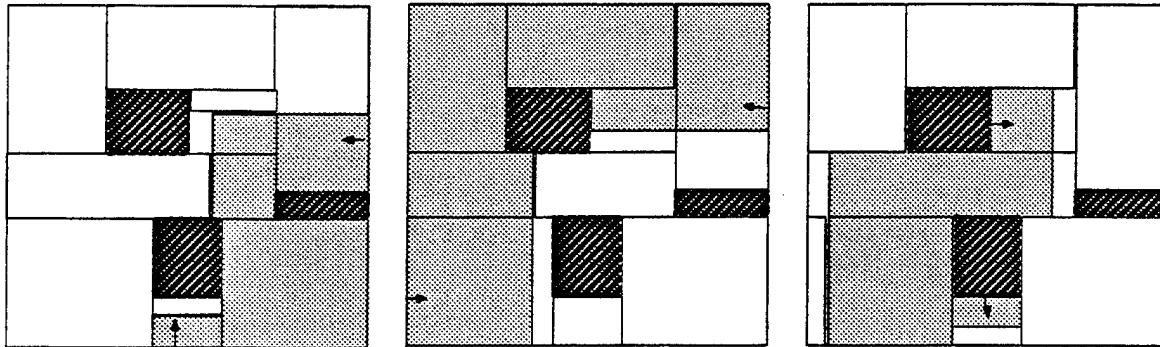
Figure 4.15: An example of cell refinement

Figure 4.16: Refined channels

4.2.5 Route Generation

When all the channels have been refined, pipe routes can be generated within the refined channels. This is done sequentially, one pipe after another. We propose two alternatives: 1) optimal route generation and 2) extreme route generation.

In optimal route generation, we use the two-pass optimality constraint propagation technique described for sequential routing. We construct the optimal route of a pipe within its refined channel. If this route overlaps channels of the pipes that have not been routed yet, the space occupied by this route is removed from these channels. We illustrate this approach using our routing example, as shown in Figure 4.17. We first generate an optimal route for pipe N_1 as shown in Figure 4.17a. This route does not overlap the channel of N_2, but does overlap the channel of N_3. Hence, the channel for N_3 is refined. The route chosen for N_1, together with the channels for N_2 and N_3, are shown in Figures 4.17b and 4.17c, respectively. Next, we find the optimal route for N_2 as shown in Figure 4.17d. This route overlaps the channel for N_3. We further refine this channel and select the optimal route for N_3 as shown in Figure 4.17e. Even though each route is optimal with respect to its refined channel, there is no guarantee of overall optimality.

Extreme route generation avoids the added cost of a second-time channel refinement by generating routes along the extreme boundary of the channels. In doing so we guarantee that the route we generate for a pipe will not overlap with the routes we
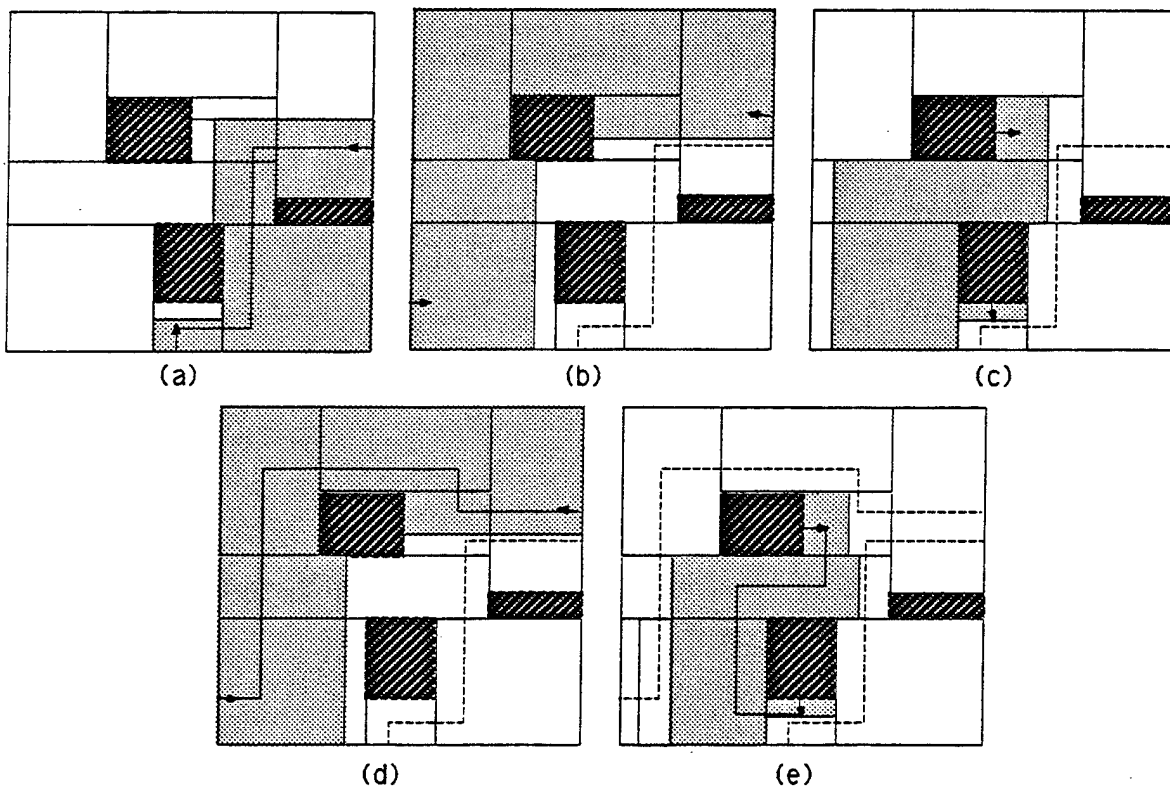
Figure 4.17: Generating optimal routes

generate for the other pipes even though they are generated independently. This is because the interaction of the pipes at these extreme locations have been considered during channel refinement. In Figure 4.18, we first generate a route for pipe N_1, then a route for pipe N_2, and finally a route for pipe N_3, all following the extreme boundary of their respective channels along $+x$ and $-y$ directions. However the routes are no longer optimal with respect to their channels.
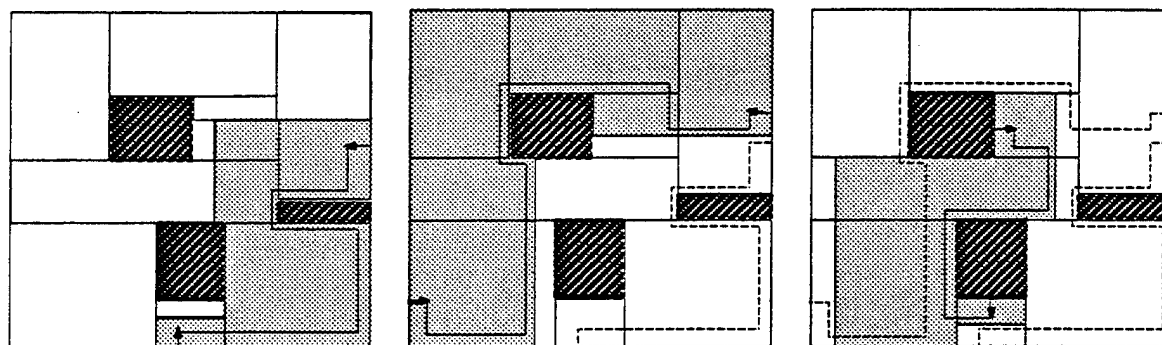
Figure 4.18: Generating extreme routes

4.2.6 Generating Alternative Channels

There are three places during the channel refinement process where a failure can occur: during constraint derivation, during boundary refinement, and during cell refinement. During constraint derivation, we may realize that no possible relative position constraint exists between a pair of pipe segments in a cell; during boundary refinement, we may discover edge capacity overflow; and during cell refinement, we may discover that the feasible region for a pipe segment in a cell no longer connects the entrance to the exit of the pipe segment. In all three cases, the failure makes explicit that the channels do not contain solutions. We must backtrack and generate alternative channels for one or more pipes, if this is possible.

As with sequential routing, the main questions here are to identify which channels to change and how to prevent looping. Similar to hierarchical path planning, failure conditions are identified and attached to each of the responsible pipes as annotations.

When we generate a new channel for a pipe, all annotations attached to it are checked to make sure that the same mistakes are not repeated.

An annotation attached to a pipe N is an expression of the form "if $[N_{i_1} : cl_1]$ and ... and $[N_{i_k} : cl_k]$ then $[N : cl]$", where the cl_k's and cl are sequences of cells. The first and the last elements of each such sequence can be a terminal which is treated as a special cell with zero size. When we search for an alternative channel for N, this annotation is used as follows: if for all $j = 1, \ldots, k$, the current channel for N_{i_j} contains the sequence of cells cl_j (or its reverse), then the channel for N must not contain the sequence cl (or its reverse). The meaning and the use of annotations are illustrated with the following examples.

We assume that the channels initially generated for the three pipes, N_1, N_2, and N_3, are $\{C_4, C_1, C_2, C_3, C_8\}$, $\{C_4, C_1, C_2, C_3, C_8, C_5\}$, and $\{C_1, C_2, C_3, C_8\}$, respectively, as shown in Figure 4.19
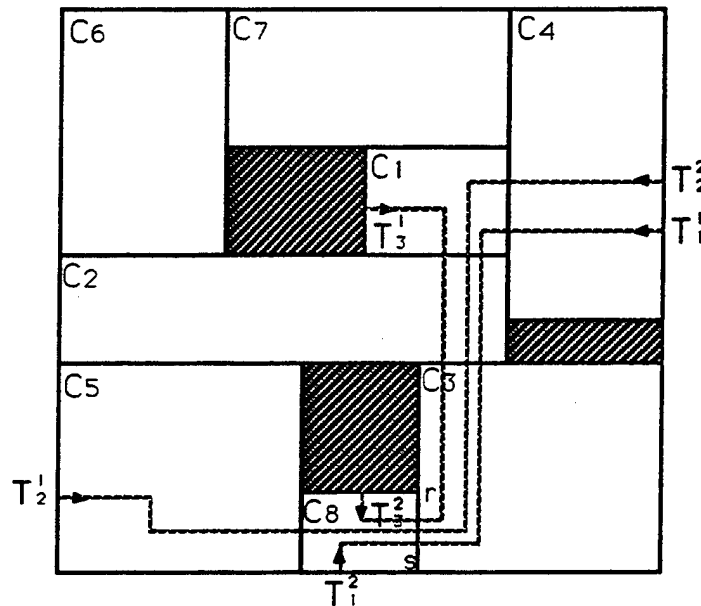

Figure 4.19: Backtracking example 1

Example 1 (Edge Capacity Overflow): During the refinement of the edge rs between cells C_3 and C_8, edge capacity overflow occurs. In the case of edge capacity

overflow, all the pipes that traverse the overloaded edge are responsible for the failure. Therefore, the pipes responsible for the failure are N_1, N_2, and N_3.

We attach the annotation "if $[N_2 : C_3, C_8]$ and $[N_3 : C_3, C_8]$ then $[N_1 : C_3, C_8]$" to the annotation list of N_1. Similarly we annotate N_2 by "if $[N_1 : C_3, C_8]$ and $[N_3 : C_3, C_8]$ then $[N_2 : C_3, C_8]$" and N_3 by "if $[N_1 : C_3, C_8]$ and $[N_2 : C_3, C_8]$ then $[N_3 : C_3, C_8]$". Assume that we decided to construct an alternative channel for N_3. When searching the connectivity graph CG, we consider its annotation list which contains: "if $[N_1 : C_3, C_8]$ and $[N_2 : C_3, C_8]$ then $[N_3 : C_3, C_8]$". Since the current channels of N_1 and N_2 both contain the cell sequence $\{C_3, C_8\}$, the new channel of N_3 must not contain the cell sequence $\{C_3, C_8\}$. Let us assume that the new channel found for N_3 is $\{C_1, C_2, C_5, C_8\}$, as shown in Figure 4.20.

The method described above for recording edge capacity overflow is very inefficient when there are many pipes traversing the overloaded edge. In this case, a better way to record the failure is to annotate the overloaded edge with its maximum capacity. When we search for a channel for a pipe, we can decide whether this pipe can traverse an edge by comparing the maximum capacity of the edge with the number of pipes that are currently traversing it (this number can be computed by looking at the pipe sets of the two adjacent cells that share this edge).

Example 2 (Cell Capacity Overflow): When we try to refine the channels using the newly generated channel for N_3, a new failure occurs during cell refinement, say, because the feasible region $FR(n_2)$ in C_8, after we subtracted $MIS(n_1, n_2)$ and $MIS(n_3, n_2)$ from it, became disconnected, i.e. it no longer connects the entrance to the exit of n_2. All the pipes in C_8, i.e. N_1, N_2, and N_3, are responsible for the failure. But this is not the case in general. Consider the example shown in Figure 4.21. In this example, there are three pipe segments, n_1, n_2, and n_3, in the cell C. The constraints among the pipe segments are $n_1 <_x n_2 \wedge n_2 <_y n_3 \wedge n_1 <_x n_3$. We assume that the x dimension of C is larger than its y dimension. A failure occurs during cell refinement when $FR(n_2)$ becomes disconnected. From the figure, it is obvious that n_3, but not n_1, is responsible for this failure. This can be discovered by the following procedure. Assume that immediately after we subtracted $MIS(n_3, n_2)$ from
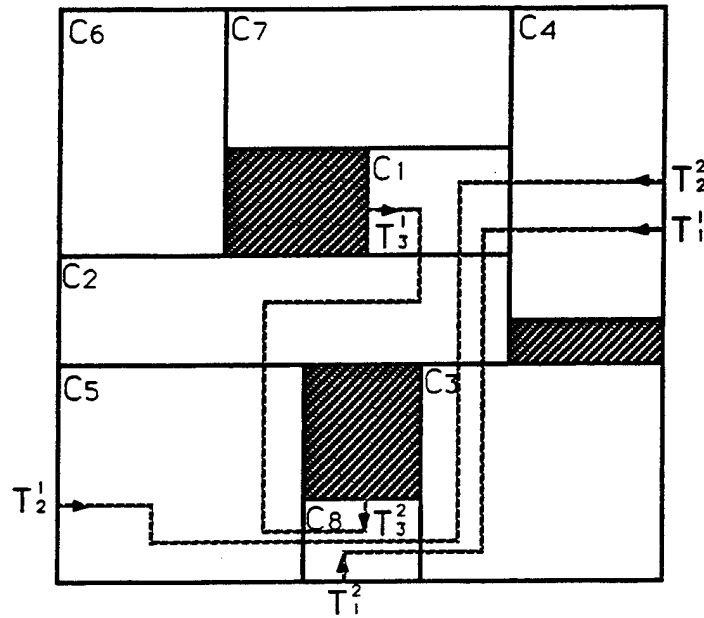
Figure 4.20: Backtracking example 2

$FR(n_2)$, $FR(n_2)$ becomes disconnected. Obviously both n_2 and n_3 are responsible for the failure. But in order to decide whether n_1 is responsible, we have to see whether n_1 is constrained by n_2 in the same way as n_2 is constrained by n_3, i.e. $n_1 <_y n_2$, or whether n_1 is constrained by n_3 in the same way as n_3 is constrained by n_2, i.e. $n_3 <_y n_1$. Since neither of these two constraints is true, n_1 is not responsible.
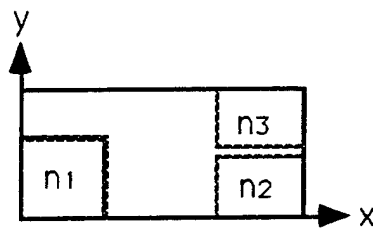

Figure 4.21: Deciding which pipes are responsible for cell refinement failure.

To record the failure which occurred in C_8, we attach the annotation "if $[N_2 : C_5, C_8, C_3]$ and $[N_3 : C_5, C_8, T_3^2]$ then $[N_1 : C_3, C_8, T_1^2]$" to the annotation list of N_1. Similarly we attach an annotation "if $[N_1 : C_3, C_8, T_1^2]$ and $[N_3 : C_5, C_8, T_3^2]$ then

$[N_2 : C_5, C_8, C_3]$" to N_2 and the annotation "if $[N_1 : C_3, C_8, T_1^2]$ and $[N_2 : C_5, C_8, C_3]$ then $[N_3 : C_5, C_8, T_3^2]$" to N_3. We assume that in backtracking we decide to re-route pipe N_2. We search CG for an alternative channel for N_2. There are two annotations in the annotation list of N_2. But only one is applicable because the condition for the other annotation is no longer true (since the channel for pipe N_1 has changed). Rather than repeatedly checking for the conditional part of an annotation to see whether it's applicable during a search, we can first create an *active annotation list* that contains all applicable annotations before the search starts. During the search we only need to refer to this active annotation list. We assume the new channel generated for N_2 is $\{C_4, C_1, C_2, C_5\}$ as shown in Figure 4.22.
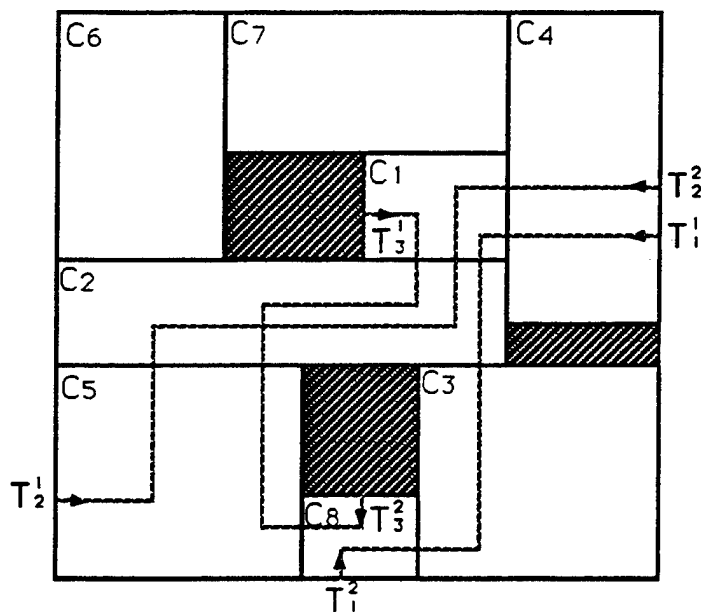

Figure 4.22: Backtracking example 3

Example 3 (Constraint Conflict): The new channel generated for N_2 yields another failure. This new failure is detected during the constraint derivation process: we fail to find a relative position constraint between the pipes N_2 and N_3 in the cell C_2 due to conflicting boundary constraints. The two pipes involved in the conflict, N_2 and N_3, are responsible for the failure.

Even though the failure is detected in C_2, the cause of the failure goes beyond C_2. In fact, the conflict is caused by the boundary constraints of the extremities of N_2 and N_3 which lie on the edge between C_2 and C_1 and the edge between C_2 and C_5. These relative boundary constraints are in turn derived from the relative position constraints in C_1 and C_5, respectively. Taking this into consideration, the annotation for N_2 is "if $[N_3 : T_3^1, C_1, C_2, C_5, C_8]$ then $[N_2 : C_4, C_1, C_2, C_5, T_2^1]$". The annotation for N_3 is completely symmetrical with the if and the then parts switched.

In general, when a constraint conflict is detected between two pipes N_i and N_j in cell C, we have to find the longest sequence of cells that are common to both channels and which contain C. We denote this cell sequence $\mathcal{CS}$ and let θ_i be the cell preceeding $\mathcal{CS}$ in the channel for N_i, and ϕ_i be the cell succeeding $\mathcal{CS}$ in the channel for N_i (θ_i and ϕ_i can be the terminals of N_i). We define θ_j and ϕ_j in a similar fashion. The annotation for N_i is "if $[N_j : \theta_j, \mathcal{CS}, \phi_j]$ then $[N_i : \theta_i, \mathcal{CS}, \phi_i]$". The annotation for N_j can be constructed symmetrically. The intuition behind the construction of this annotation is that the cause of the failure is not where the failure is detected, but rather where the two channels start to "cross" and where the crossing ends.

We assume that we re-route N_2. Hence we search CG for an alternative channel for N_2. At this point, there are three annotations attached to N_2, but only two of them are active. The active annotation list of N_2 is $\{[N_2 : C_3, C_8, C_5], [N_2 : C_4, C_1, C_2, C_5, T_2^1]\}$ (the conditional parts of these annotations are not kept in the active list because they have already been checked). The search resulted in a channel $\{C_4, C_1, C_7, C_6, C_2, C_5\}$ for pipe N_2. As discussed in the previous subsections, this channel for N_2 together with the other two channels for N_1 and N_3 contain solutions.

This backtracking procedure has the following property:

Property 4: This backtracking procedure is complete in the sense that if there exists an assignment of channels to pipes such that they contain solution routes for all of the pipes, this procedure will find such an assignment.

This property comes from the fact that the recorded failure conditions are the weakest conditions that will prevent the failure from re-occurring and hence they will
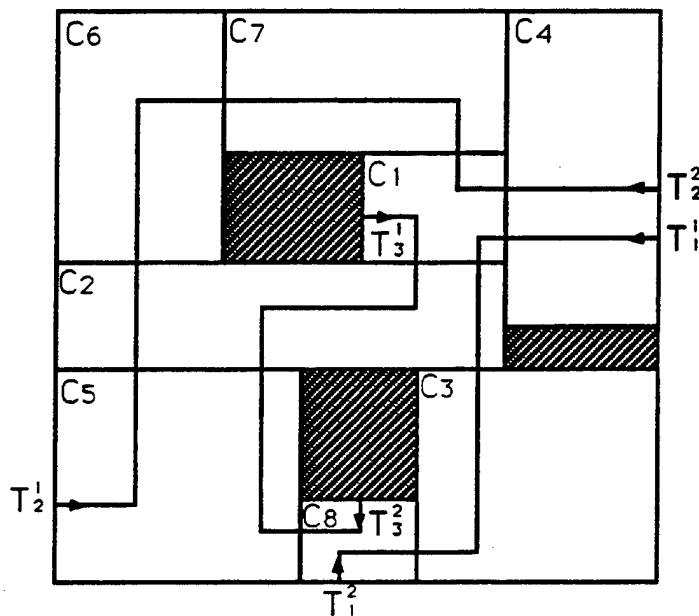
Figure 4.23: Successful channels

not exclude any feasible solution.

4.3 Extension to Three-Dimensional Case

In this section we describe the extension of the two-dimensional parallel routing algorithm developed in the previous section to the three-dimensional case. The decomposition of the algorithm into its basic components, i.e. channel generation, constraint derivation, channel refinement, route generation, and backtracking remains the same, but each of these components is modified to some extent.

The most important change occurs in the constraint derivation step. This is due to the fact that the third dimension allows two pipe routes to "cross" each other, i.e. the relative position of two pipe segments in a cell is no longer uniquely determined by the relative boundary constraints on the extremities of these two segments. Constraint derivation becomes a search for an assignment of a relative position constraint for every pair of pipe segments in a cell.

This additional level of search has a significant impact on the backtracking algorithm. In the two-dimensional case, if a failure occurs during channel refinement, we backtrack and generate alternative channels for some pipes. In the three-dimensional case, since there can be more than one possible relative position constraint between two pipe segments, when a failure occurs during channel refinement, we should first try to generate alternative relative position constraints for one or more pair of pipe segments. Only when the algorithm fails to find a relative position constraint for a pair of pipe segments, does it backtrack and generate alternative channels.

4.3.1 Channel Generation

The channel generation step is just a straightforward generalization of the two-dimensional case.

4.3.2 Constraint Derivation

As in the two-dimensional case, in order to refine the channel of a pipe N, we must determine the position of the pipe segment of N relative to the other pipe segment(s) in each of the cells along the channel that are traversed by more than one pipe. The constraint language used in the two-dimensional case is augmented by adding another predicate, "$<_z$", to describe the relative position along the third dimension.

In the two-dimensional case, we can uniquely determine the relative position between two pipe segments, if there exists a relative boundary constraint between one pair of extremities of these two pipe segments. But in the three-dimensional case, this is no longer true as illustrated by the examples in Figure 4.24. In this figure, pipe segment n_1 enters through the top surface of a cell C and exits through its front, while n_2 enters on the right and exits through the front. Although we can derive the relative boundary constraint $p_1 <_y p_2 \wedge p_2 <_z p_1$ between the extremities p_1 of n_1 and p_2 of n_2, there is no unique relative position between n_1 and n_2. (Figure 4.24 shows four possible relative positions between n_1 and n_2.) To proceed, we must choose one of the possible relative positions of n_1 and n_2.
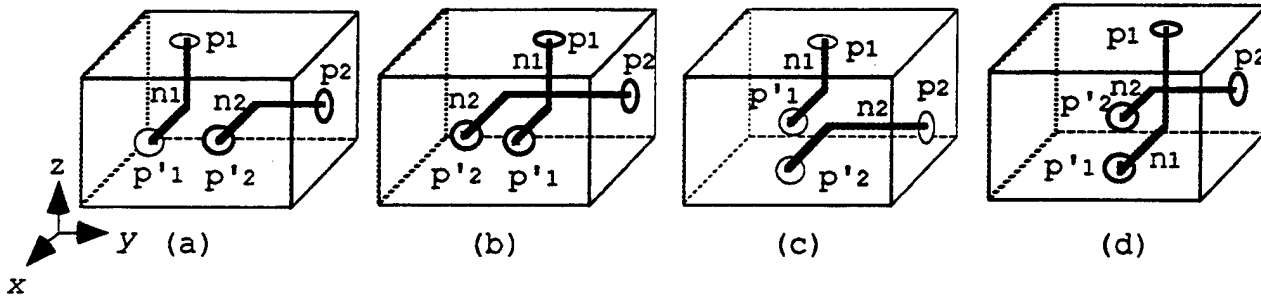
Figure 4.24: Several possible relative positions between two pipe segments

There are several factors affecting this choice:

- The geometry of the cell C and the existing relative position constraints among the pipe segments in C.

- The geometry of the faces of C containing the extremities of n_1 and n_2, and the existing relative boundary constraints among the extremities on these faces.

- The existing relative position constraints between the corresponding pipe segments in the adjacent cells.

First of all, the choice of a relative position constraint between n_1 and n_2 is affected by the geometry of C and the existing relative position constraints among the pipe segments in C. For example, if the y dimension of C is larger than its z dimension, then the relative position constraints shown in Figures 4.24a and 4.24b are preferred over those shown in Figures 4.24c and 4.24d (assuming that the x dimension is not relevant). In another example shown in Figure 4.25, there is another pipe segment n_3 in C in addition to n_1 and n_2, and there is an existing relative position constraint $n_3 <_y n_1$. In this case, the constraint $n_1 <_y n_2$ (Figure 4.25a) is less desirable than $n_2 <_z n_1$ (Figure 4.25b), unless the y dimension of C is larger than its z dimension. When there are more than two pipe segments in a cell, we must make sure that we do not derive inconsistent relative position constraints among these pipe segments. For example, if we have three pipe segments, n_1, n_2, and n_3, in a cell C, and there

are existing relative position constraints $n_1 <_x n_2 \wedge n_2 <_x n_3$, the relative position constraint between n_1 and n_3 must not contain $n_3 <_x n_1$. We call the derivation of $n_3 <_x n_1$ a *relative position constraint violation*.
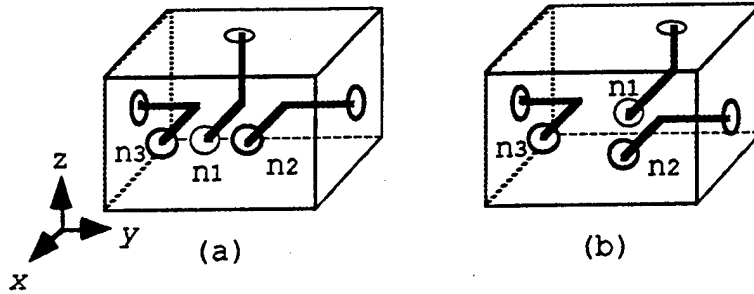

Figure 4.25: Affecting the choice of relative position constraint

Secondly, the choice of an appropriate relative position between n_1 and n_2 is affected by the geometry of the faces of the cell containing the extremities of n_1 and n_2. This is because we can derive a relative boundary constraint from a relative position constraint. For example, in Figure 4.24a, we can derive the relative boundary constraint $p_1' <_y p_2'$ where p_1' and p_2' are the extremities of n_1 and n_2 respectively. If the y dimension of the face containing p_1' and p_2' is smaller than its z dimension, then the relative position constraint $n_2 <_z n_1$ is preferred over $n_1 <_y n_2$, because $p_2' <_z p_1'$ is preferred over $p_1' <_y p_2'$. If p_1' and p_2' share the face with other extremities of pipe segments, then the existing relative boundary constraints among these extremities also affect the choice.

Lastly, the appropriate choice of a relative position constraint between n_1 and n_2 may also depend on the relative position constraints between the corresponding pipe segments in the adjacent cells. For example, in Figure 4.26, there are two cells C_1 and C_2 and two pipes N_1 and N_2, both of which traverse C_1 and C_2. There are four possible relative position constraints between n_1 and n_2 in C_1 which are shown in Figures 4.27a - 4.27d. There are also four possible relative position constraints between n_1 and n_2 in C_2 which are shown in Figures 4.27e - 4.27h. The relative positions in C_1 and C_2 shown in each column are more compatible with each other
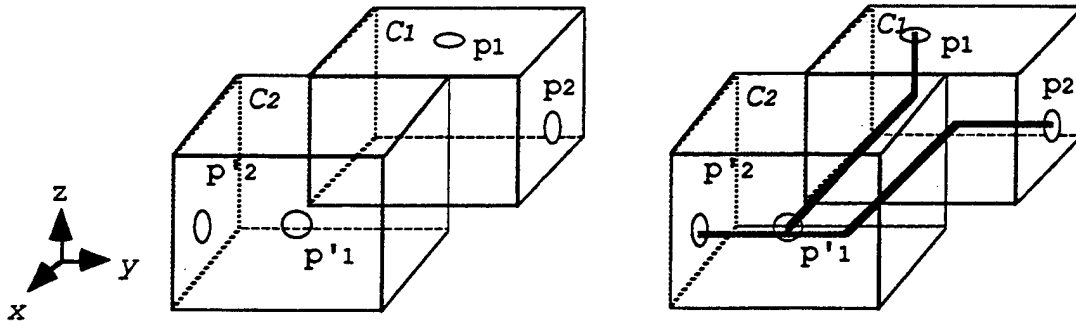
Figure 4.26: Relative position constraints between two pipe segments in two adjacent cells

because they both project the same relative boundary constraint on the face that is at the intersection between C_1 and C_2. On the other hand, The relative position constraints shown in Figures 4.27a and 4.27f are not compatible because they project conflicting relative boundary constraints on the face at the intersection between C_1 and C_2. We call this a *relative boundary constraint violation*.
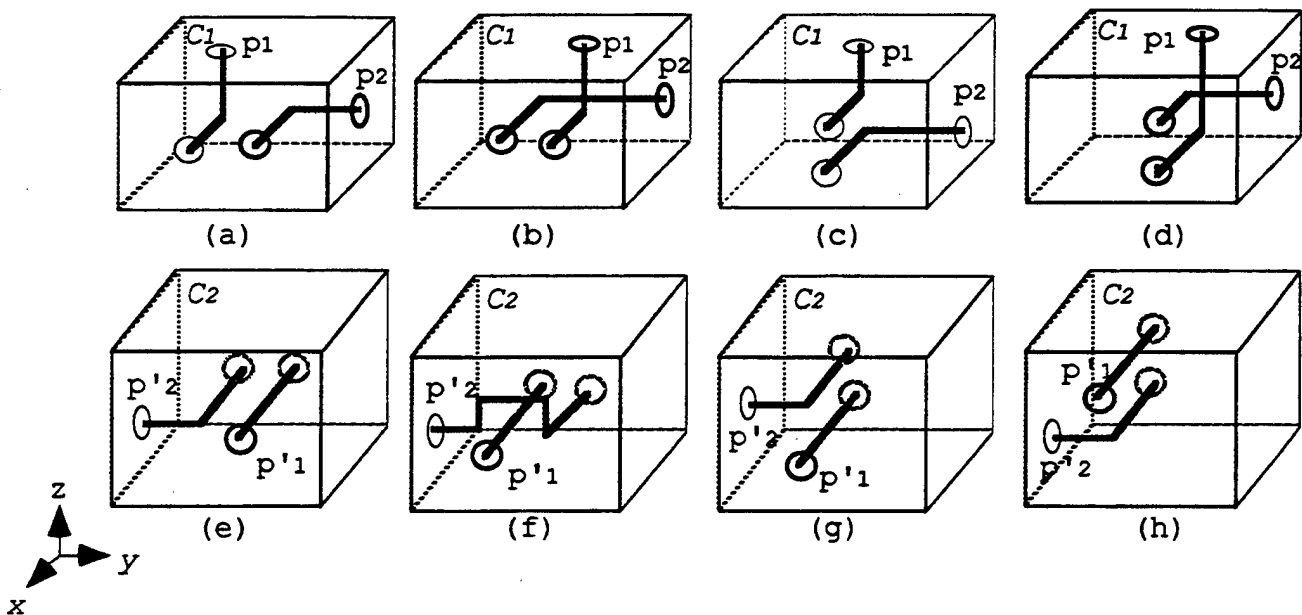
Figure 4.27: Several possible relative positions between two pipe segments in C_1 and C_2

Therefore, constraint derivation is aimed at choosing the appropriate relative position of every pair of pipe segments in every cell that is traversed by more than one pipe, taking into consideration the factors discussed above. It can be organized as the search of a tree in which each *level* (except the root) corresponds to a pair of pipe segments in a cell. Each *node* of this tree represents a relative position constraint between two pipe segments. Figure 4.28 provides a simple example of a tree involving two cells, C_1 and C_2, and three pipes, N_1, N_2, and N_3. All three pipes traverse C_1 and only pipes N_1 and N_2 traverse C_2. The search tree has four levels (in addition to the root). The top three levels correspond to the three pairs of pipe segments in C_1, and the bottom level corresponds to the pair of pipe segments in C_2. For simplicity, only a subset of the nodes is shown in the figure. The path from the root node to the only leaf node shown in the figure corresponds to a feasible assignment of a relative position constraint to every pair of pipe segments. This assignment is depicted in the drawing above the tree. The three factors that we discussed above can be expressed in the form of a cost function evaluated at every traversed node. In particular, a node can not be traversed if it corresponds to either a *relative position constraint violation* or a *relative boundary constraint violation*.

As in the two-dimensional case, a relative position constraint between two pipe segments, n_1 and n_2, can be used to determine relative boundary constraints between the extremities of these two pipe segments. This relative boundary constraint in turn can be used to constrain the choice of the relative position constraint between the corresponding pipe segments in the adjacent cell.

If we succeed in finding a path leading from the root node to a leaf node of the tree, this path corresponds to an assignment of a relative position constraint for every pair of pipe segments. We can then proceed to the channel refinement phase. If we fail to find a path, we must backtrack to find alternative channels for some of the pipes.
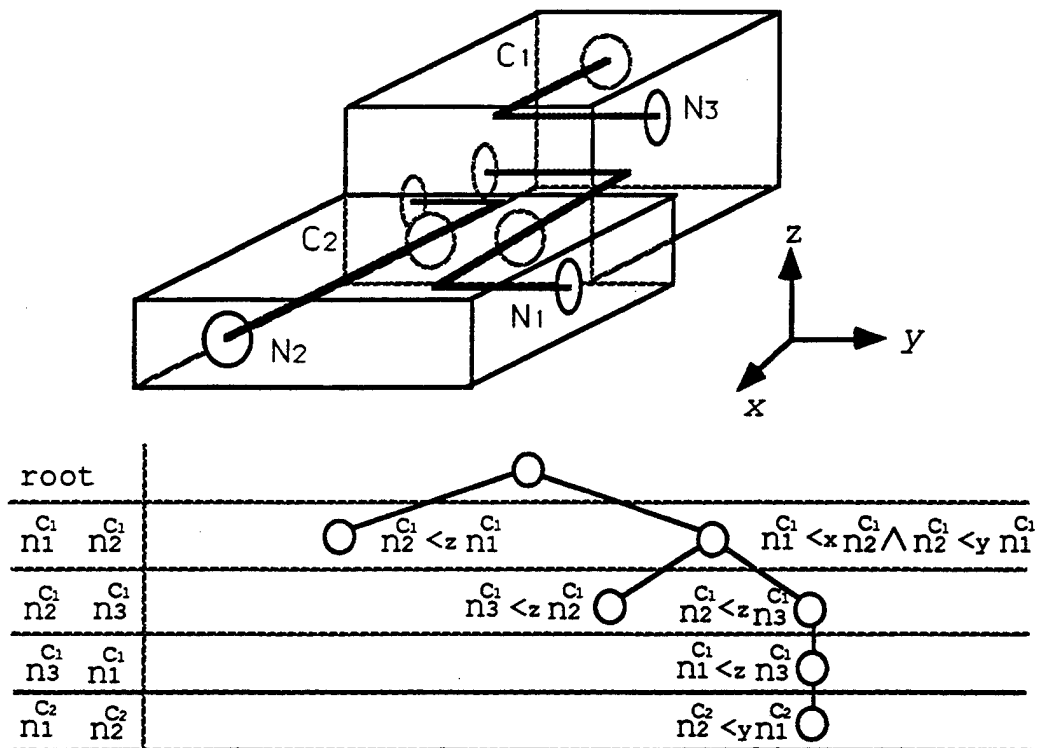
Figure 4.28: Searching for relative position constraints

4.3.3 Channel Refinement

Channel refinement is carried out in two steps: *boundary refinement* and *cell refinement.*

a. Boundary Refinement We use the derived relative boundary constraints to refine the entrances and exits of the pipe segments in a cell. Let S be a rectangular face traversed by l pipes. Let $\{p_1,\ldots,p_l\}$ be the extremities of the corresponding pipe segments in S. Without loss of generality, we assume that S is parallel to the xy plane and is described by four parameters $[x^a, x^b, y^a, y^b]$. The subset of S that p_i is allowed to lie in is another rectangular region described by $[x_i^a, x_i^b, y_i^a, y_i^b]$ which we call the feasible face of p_i. The goal is to refine the feasible face of p_i (starting with S) such that if the pipe segment n_i traverses S within its refined feasible face, it will not prevent other pipe segments from being routed through S. As in the two-dimensional case, this is accomplished in two steps: 1) creating sorted lists of the extremities along x and y axes; 2) for each of the extremities, refining its feasible face based on its position in the sorted lists.

Since the refinement along the x and y dimensions can be carried out independently, we only describe the algorithm for refining the x dimension. We first sort the extremity list along the x axis using the relative boundary constraints. Unlike in the two-dimensional case, there can be more than one sorted list, as illustrated in Figure 4.29. Let $\{p_{k_1},\ldots,p_{k_s}\}$, where $s \leq l$ and $k_i \in [1,\ldots,l]$, be a sorted list of extremities along the x axis. We can refine the x interval of the feasible face of p_{k_j} by letting $x_{k_j}^a = x^a + 2r * (j - 1)$ and $x_{k_j}^b = x^b - 2r * (s - j)$. A self-explanatory example illustrating this refinement process is given in Figure 4.29.

b. Failure in Boundary Refinement During the boundary refinement process, we may discover that the feasible face of one or more extremities on a face S has been reduced to the empty set (e.g. $x_i^a \geq x_i^b$). We call this failure *face capacity overflow.* There are two possible cases to be considered: 1) the failure is caused by too many pipes traversing S and hence is independent of the boundary constraints;

boundary constraints:

p1 <x p2 ∧ p1 <x p3 ∧ p4 <x p2 ∧ p4 <x p3 ∧
p2 <y p3 ∧ p2 <y p1 ∧ p4 <y p3 ∧ p4 <y p1
sorted lists (along x-axis):
{p1, p3} {p4, p2} {p1, p2} {p4, p3}
sorted lists (along y-axis):
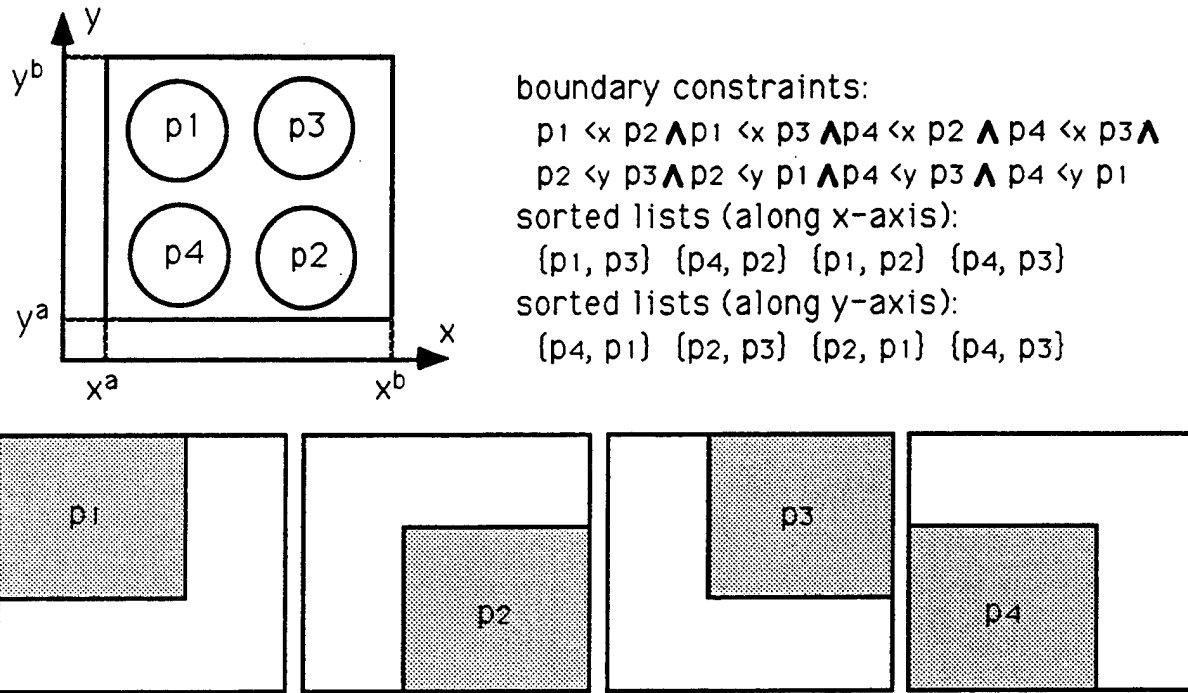{p4, p1} {p2, p3} {p2, p1} {p4, p3}

Figure 4.29: Boundary refinement

2) the failure is caused by inappropriate boundary constraints among the extremities on S. When a face capacity overflow occurs, we must first determine which case it is, and then discover the failure condition and record it by attaching an appropriate annotation to the face.

There is a simple algorithm for detecting the first case of boundary refinement failure. From the size of S, we can calculate the maximum number of pipes that can go through this face. Letting the pipe radius be r, the maximum number of pipes that can go through S is $[floor((x^b - x^a)/(2r)) + 1] * [floor((y^b - y^a)/(2r)) + 1]$, where $floor(x)$ evaluates to the largest integer less than x. Hence, if the number of pipes that traverse S is larger than this number, we know that the failure is of case 1 and we must backtrack to generate alternative channels for some of the pipes that traverse S.

If the number of pipes traversing S is less than the maximum capacity, then we know that the failure is caused by some inappropriate boundary constraints in S.

The problem is to find out the responsible boundary constraints. Let us assume that p_k's feasible face has been reduced to the empty set along the x-axis (i.e. $x_k^a \geq x_k^b$). Then all of the extremities that belong to the same sorted list (along x axis) as p_k are responsible for the failure (if there are more than one sorted list that contains p_k, we choose the longest list). The conjunction of the relative boundary constraints (along the x-axis) applied to these extremities should not appear together again and thus is added as an annotation to the annotation list of the face S in which p_k lies. This can be best illustrated by the example shown in Figure 4.30. In this example, the extremities p_1, p_2, p_3, and p_4 on face S are constrained as follow: $p_1 <_x p_2 \wedge p_1 <_x p_3 \wedge p_3 <_x p_4 \wedge p_1 <_x p_4 \wedge p_2 <_y p_3 \wedge p_2 <_y p_4$. When we detected that the feasible surface for p_1 is empty along the x-axis, we found two sorted lists along the x-axis that contain p_1. The longer of the two is $\{p_1, p_3, p_4\}$. Hence the failure condition is the conjunction $p_1 <_x p_3 \wedge p_3 <_x p_4$, which we then attach to the annotation list of S.
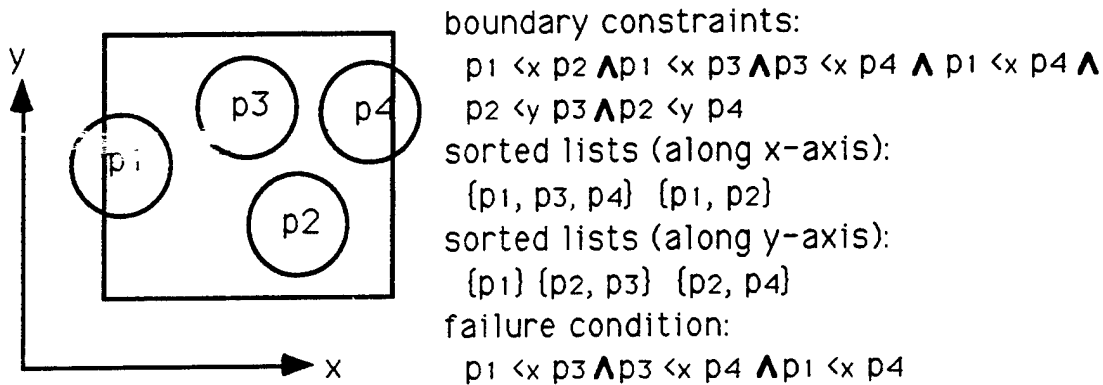

boundary constraints:
  p1 <x p2 ∧p1 <x p3 ∧p3 <x p4 ∧ p1 <x p4 ∧
  p2 <y p3 ∧p2 <y p4
sorted lists (along x-axis):
  {p1, p3, p4} {p1, p2}
sorted lists (along y-axis):
  {p1} {p2, p3} {p2, p4}
failure condition:
  p1 <x p3 ∧p3 <x p4 ∧p1 <x p4

Figure 4.30: Boundary-refinement failure condition.

c. Cell Refinement   Assume that the boundary refinement process has been successful. We now proceed to cell refinement. The algorithm for cell refinement is similar to that described for the two-dimensional case. The only difference is in the computation of the Minimum Impact Solution.

In computing $MIS(n_j, n_i)$, there are three cases to consider as illustrated in Figure 4.31. The first case corresponds to when n_j and n_i are constrained to each other along

one axis (e.g. x-axis), the second case corresponds to when n_j and n_i are constrained to each other along two axes (e.g. x and y axes), and the third case corresponds to when n_j and n_i are constrained to each other along all three axes. Figure 4.31 illustrates the construction of $MIS(n_j, n_i)$ for the simple situation where $FR(n_j)$ is a single rectangloid. In the case where $FR(n_j)$ consists of more than one rectangloid, the two-step process described for the two-dimensional case also applies here.

d. Failure in Cell Refinement During cell refinement we may discover that the feasible region of a pipe segment n_i, $FR(n_i)$, no longer connects the entrance to the exit of n_i in a cell C. Let us assume that $FR(n_i)$ became disconnected immediately after $MIS(n_j, n_i)$ was subtracted from it. We construct a conjunction, $\mathcal{L}$, of relative position constraints to record this failure condition as follow. $\mathcal{L}$ is set to empty initially. For each atomic relative position constraint which exists between n_i and n_j, say $n_i <_x n_j$, we add it to $\mathcal{L}$. In addition, for every pipe segment n in C, if $n <_x n_i$ or $n_j <_x n$, then we also add this atomic constraint to $\mathcal{L}$. The same can be done if $n_j <_x n_i$, $n_i <_y n_j$, $n_j <_y n_i$, $n_i <_z n_j$, or $n_j <_z n_i$ exists between n_i and n_j. This process is illustrated with the example shown in Figure 4.32.

In this example, there are four pipe segments, n_1, n_2, n_3, and n_4, in the cell C. The relative position constraints among these pipe segments are given in the figure. Let us assume that we discovered cell refinement failure after we refined the feasible region of n_2 by computing $FR(n_2) - MIS(n_3, n_2)$. Since $n_3 <_x n_2$ and $n_2 <_y n_3$, $\mathcal{L} = n_3 <_x n_2 \wedge n_2 <_x n_1 \wedge n_2 <_y n_3 \wedge n_1 <_y n_2$. We attached $\mathcal{L}$ to C to record the failure condition.

4.3.4 Route Generation

If the channel refinement is successful, we can proceed to generate a route within the refined channel of each pipe. The route generation is just a straightforward generalization from the two-dimensional case.
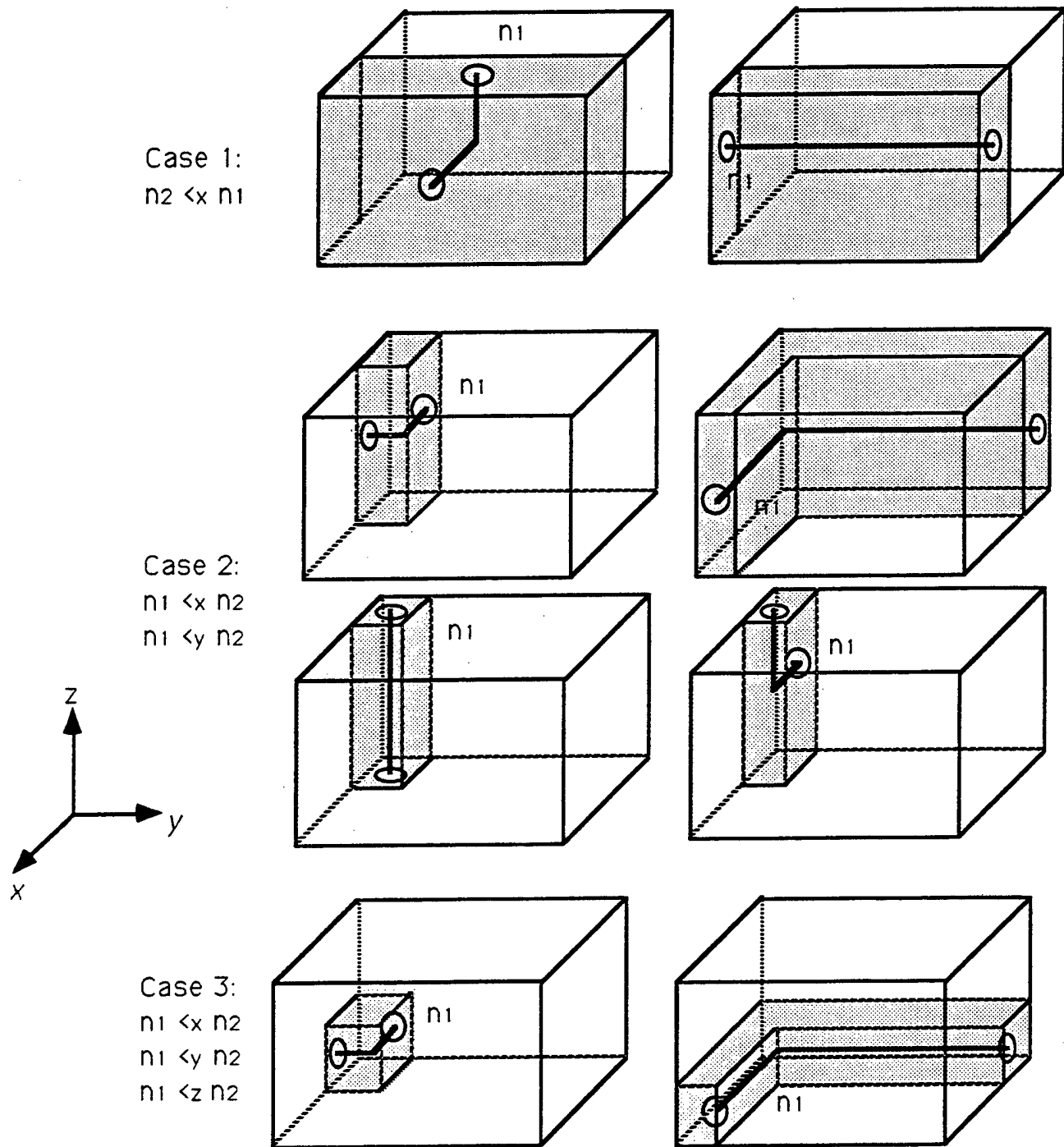
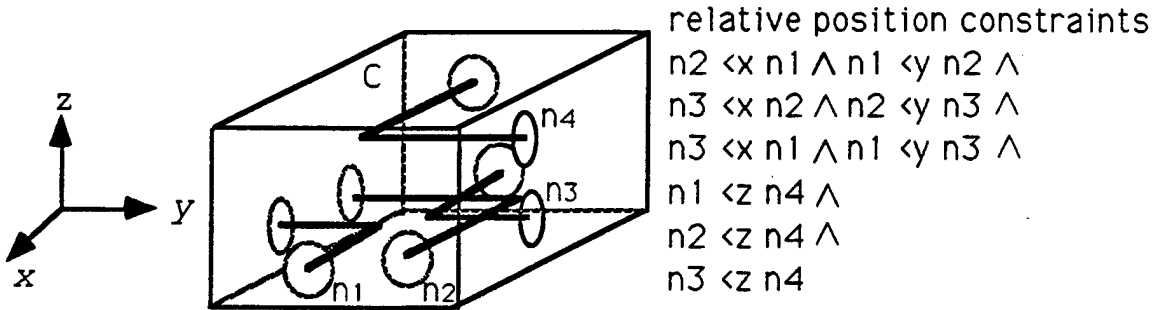Figure 4.31: Computing Minimum Impact Solution in the three-dimensional case

relative position constraints
n2 <x n1 ∧ n1 <y n2 ∧
n3 <x n2 ∧ n2 <y n3 ∧
n3 <x n1 ∧ n1 <y n3 ∧
n1 <z n4 ∧
n2 <z n4 ∧
n3 <z n4

Figure 4.32: Condition for cell refinement failure

4.3.5 Backtrack to Generate Alternative Relative Position Constraints

As discussed in the previous subsection, a failure can occur during channel refinement. Unlike in the two-dimensional case, channel refinement failure does not necessary mean that the channels for some of the pipes are inappropriate. In the three-dimensional case, channel refinement failure can be caused by inappropriate relative position constraints among some pipe segments. Hence, we first backtrack and generate alternative relative position constraints for some of the pipe segments. In order to prevent the same mistakes from occurring again, we incorporate the annotation recording the failure condition into the tree search. We describe how this is done for the boundary refinement failure case and for the cell refinement failure case.

a. Boundary Refinement Failure In the case of boundary refinement failure, there are two possibilities: 1) the face area is too small for the pipes to go through independent of the relative boundary constraints; 2) the face is too small for the pipes to go through because of the inappropriate relative boundary constraints among some of the extremities on the face. In the first case we need to backtrack to generate alternative channels. In the second case we first try to generate alternative relative position constraints for some of the pipe segments as discussed below.

Let us assume that an annotation recording the failure is $p_{i_1} <_w p_{j_1} \wedge \ldots \wedge p_{i_q} <_w p_{j_q}$ where w can represent x, y, or z-axis, $i_k \in [1, \ldots, t]$ and $j_k \in [1, \ldots, t]$. We

assume that the relative boundary constraint $p_{i_k} <_w p_{j_k}$ is derived from the relative position constraint $n_{i_k} <_w n_{j_k}$ in one of the adjacent cells. We also assume that the node corresponding to pipe segment pair n_{i_k} and n_{j_k} is an ancestor of the node corresponding to pipe segment pair n_{i_l} and n_{j_l} if $k < l$. This annotation is incorporated into the tree search by increasing the cost of traversing a node corresponding to the relative position constraint $n_{i_q} <_w n_{j_q}$ if, together with its ancester nodes, it implies the relative boundary constraints recorded by the annotation.

b. Cell Refinement Failure In the case of cell refinement failure, let us assume that an annotation recording the failure is $n_{i_1} <_w n_{j_1} \wedge \ldots \wedge n_{i_q} <_w n_{j_q}$ where w can represent x, y, or z-axis, $i_k \in [1, \ldots, t]$ and $j_k \in [1, \ldots, t]$. We also assume that the node corresponding to pipe segment pair n_{i_k} and n_{j_k} is an ancestor of the node corresponding to pipe segment pair n_{i_l} and n_{j_l} if $k < l$. To incorporate this annotation into the tree search, we find all the nodes of the tree that correspond to $n_{i_q} <_w n_{j_q}$. For each of these nodes, if it and its ancestor nodes together contain the relative position constraint recorded by the annotation, then we delete this node or its descendants (if any) from the openlist and we mark this node"infertile"[2]. After modifying the tree, we search the tree for alternative relative position constraints. Of course, when we are about to expand a node, we must first check to make sure that the relative position constraint corresponding to this node together with those corresponding to its ancestor nodes do not violate any relative position constraint annotation attached to the cell. If it violates an annotation, this node will not be expanded and will be marked "infertile" (Recall from the discussion in subsection 4.3.2 that there are two other cases where a node cannot be expanded: relative position constraint violation and relative boundary constraint violation. In either of these two cases, we also mark this node "infertile").

We illustrate this process with the example shown in Figure 4.33. In this example, there are two cells, C_1 and C_2, and four pipes, N_1, N_2, N_3, and N_4. Pipes N_1, N_2, and N_3 traverse C_1 and pipes N_1, N_2, and N_4 traverse C_2. To simplify the presentation, we

---

[2]This marking has no meaning right now but is used later for the further pruning of the tree and for discovering channel failure conditions.
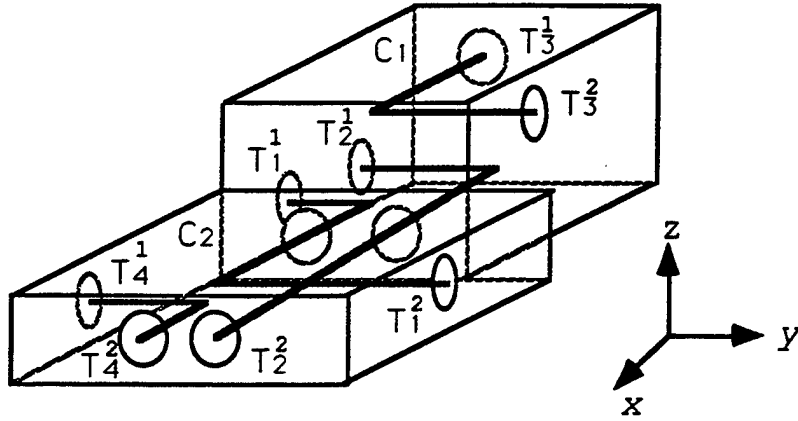
Figure 4.33: An example of cell refinement failure.

assume that in C_1 the only feasible relative position between N_1 and N_3 and between N_2 and N_3 are $n_1^{C_1} <_z n_3^{C_1}$ and $n_2^{C_1} <_z n_3^{C_1}$, respectively. But we allow three possible relative position constraints between N_1 and N_2, i.e. $n_1^{C_1} <_z n_2^{C_1}$, $n_2^{C_1} <_z n_1^{C_1}$, and $n_2^{C_1} <_x n_1^{C_1} \wedge n_1^{C_1} <_y n_2^{C_1}$. In C_2, three possible relative position constraints exist between N_1 and N_2, i.e. $n_1^{C_2} <_z n_2^{C_2}$, $n_2^{C_2} <_z n_1^{C_2}$, and $n_1^{C_2} <_x n_2^{C_2} \wedge n_2^{C_2} <_y n_1^{C_2}$. We assume that there are also three possible relative position constraints between $n_1^{C_2}$ and $n_4^{C_2}$ and between $n_2^{C_2}$ and $n_4^{C_2}$. But since these constraints are not relevant to the discussion in this example, they are not made explicit in the Figure.

We first found a path from the root node to a leaf node l_1 as shown in Figure 4.34a. When we refine the cell C_1 using the relative position constraints derived from this path, a failure occurs and the annotation recording the failure is "$n_2^{C_1} <_z n_1^{C_1} \wedge n_1^{C_1} <_z n_3^{C_1}$." Using the procedure described above, we delete from the openlist all descendants (shown grey) of the node corresponds to $n_1^{C_1} <_z n_3^{C_1}$, and we mark this node "infertile" (shown black). The nodes with a dot in the center are the other explored nodes. When we search this tree again for an alternative path, we find a path leading to a leaf node t_2 as shown in Figure 4.34b. Again, a failure occurs when we refine the cell C_1. This time the annotation recording the failure is "$n_1^{C_1} <_z n_2^{C_1} \wedge n_2^{C_1} <_z n_3^{C_1}$." We delete all descendants of the node corresponding to $n_2^{C_1} <_z n_3^{C_1}$ and mark this node "infertile". We continue the search and find a path leading to a leaf node t_3 as shown in Figure 4.34c. This time a failure occurs during
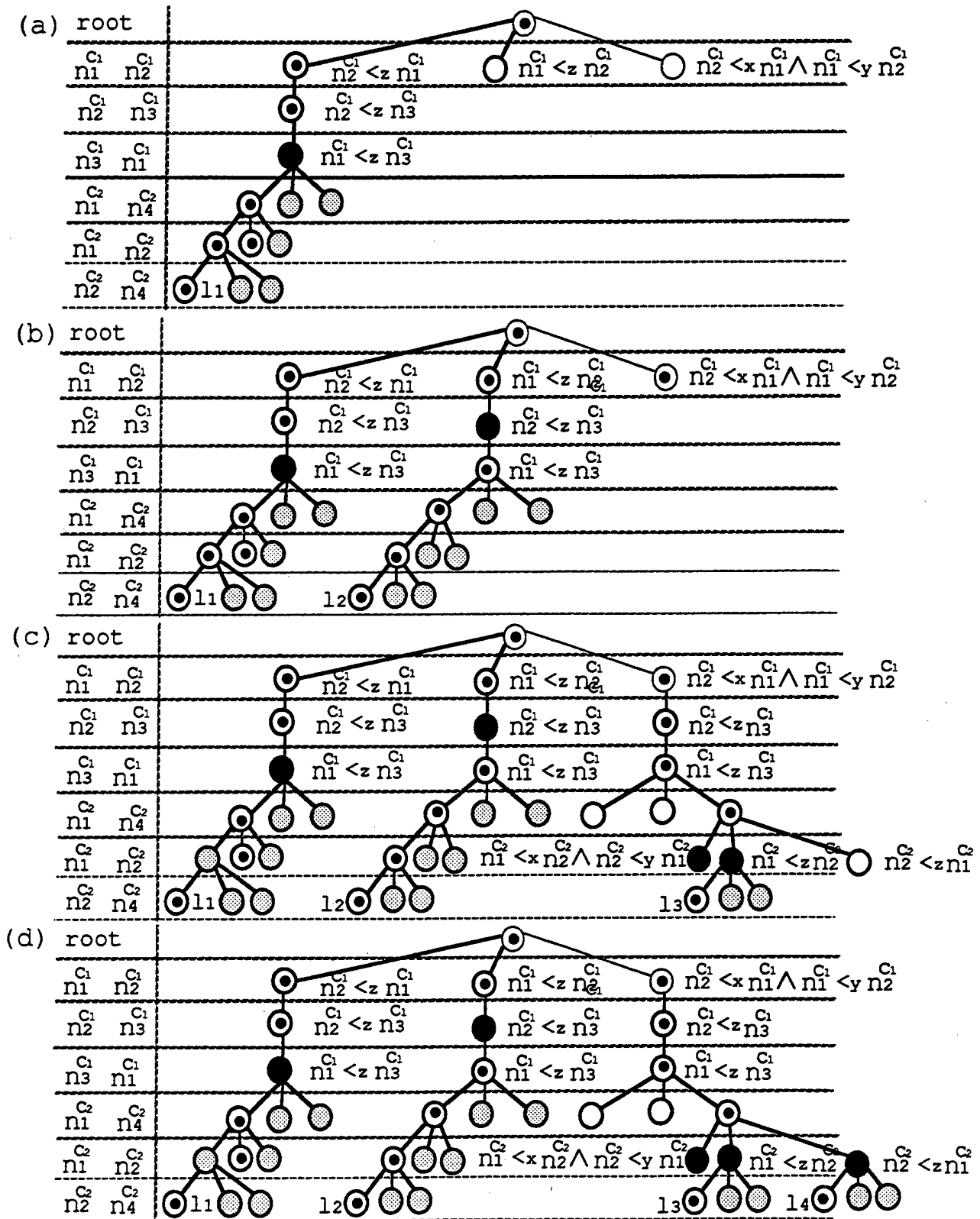
Figure 4.34: Modifying the search tree to record cell refinement failures

the refinement of C_2. The annotation recording this failure is $n_1^{C_2} <_z n_2^{C_2}$. Notice that in this figure, the node corresponding to $n_1^{C_2} <_x n_2^{C_2} \wedge n_2^{C_2} <_y n_1^{C_2}$ is shown black because it violates the relative boundary constraint derived from its ancester node which corresponds to $n_2^{C_1} <_x n_1^{C_1} \wedge n_1^{C_1} <_y n_2^{C_1}$. The search continued and we find a path leading to a leaf node t_4 as shown in Figure 4.34d. But again a failure occurs during refinement of C_2. The annotation recording this failure is $n_2^{C_2} <_z n_1^{C_2}$.

By incorporating the annotations into the search tree, the procedure illustrated above prevents the same failure from occurring again. However, further pruning of the tree is possible when we detect that all children of a node are marked "infertile". We will continue with the previous example to illustrate this process. We reproduce the tree of Figure 4.34d in Figure 4.35 and label some of the relevant nodes. Notice that nodes t_{12}, t_{13}, and t_{14} are all "infertile" and they are the only children of node t_{11}. As we have discussed above, t_{13} and t_{14} are themselves responsible for being "infertile" while t_6 as well as t_{12} are responsible for t_{12} being "infertile". We delete all descendants of t_6 from the openlist and mark it "infertile" because we know that all of t_6's descendants at same level as t_{12} are infertile. For example, if we expand t_9 or t_{10}, their children would be "infertile". Let us now look at node t_3 which is the only child node of t_2. t_3 is infertile. We identify that t_1 is the only other node that is responsible. We mark t_1 "infertile". Unfortunately, no additional pruning of the tree is possible because none of t_1's descendants is in the openlist. Similarly, t_5 is the only child of t_4 and is "infertile". We identify that t_4 is responsible. Therefore we mark t_4 "infertile" as well. Again no further pruning of the tree is possible. In fact, at this point the openlist is empty which indicates that the search has failed and we need to backtrack to generate alternative channels.

4.3.6 Backtrack to Generate Alternative Channels

As discussed in the previous subsection, there are two cases where we need to backtrack and generate alternative channels: 1) boundary refinement failure; and 2) constraint derivation failure.
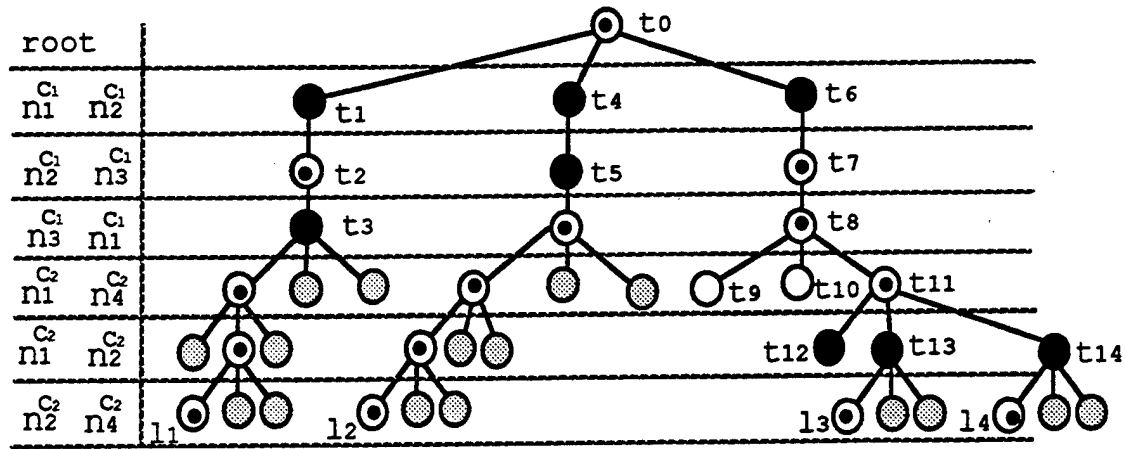
Figure 4.35: Further trimming of the search tree

a. Boundary Refinement Failure   As described in Subsection 4.3.3, one case of boundary refinement failure is due to too many pipes traversing a face. This failure case is independent of the relative boundary constraints among the extremities on the face and thus is independent of the relative position constraints among the pipe segments in the cell. In this case, we need to backtrack and generate alternative channels for some of these pipes. The recording of this failure condition for the purpose of generating alternative channels is the same as that in the two-dimensional case. Let us assume that this failure occurs in face S between cells C_k and C_l and that S contains pipe extremities $p_1, \ldots, p_n$, where p_j is the extremity of pipe N_j on S. We attach the annotation "if $[N_j : C_k, C_l]$ for $j = 1, \ldots, n$ and $j \neq i$ then $[N_i : C_k, C_l]$" to every pipe N_i, $i = 1, \ldots, n$. These annotations will guarantee that these pipes will not all go through the same S again when alternative channels are generated.

b. Constraint Derivation Failure   Constraint derivation failure occurs when the *openlist* becomes empty during the search of the tree for alternative position constraints. When this occurs, we need to backtrack and generate alternative channels for some of the pipes. To discover the cause of failure we use the "infertile" markings that we have made during alternative relative position constraint generation. This annotation construction process is carried out in two steps. In the first step, we examine each of the cells that contain more than one pipe segment to determine

whether any of the pipes within this cell is responsible for the failure, and if so, generate annotations locally within that cell for the responsible pipes. In the second step, we merge annotations that we generated in the first steps, as these annotations are usually too "strong". We describe these two steps in detail below.

Let C be a cell that contains pipe segments $n_1, \ldots, n_l$ for $l > 1$. For each pair of pipe segments, n_j and n_k, we examine all the parents of the nodes corresponding to the pipe segment pair n_j and n_k. If we can find a parent node such that all its children nodes are marked "infertile", we know that pipes N_j and N_k, which correspond to n_j and n_k in C, respectively, are responsible for the failure. For example in Figure 4.35, we first look at pipe segment pair $n_1^{C_1}$ and $n_2^{C_1}$. There is only one parent node which is the root node. Since all three children of the root are marked "infertile", we determine that N_1 and N_2 are responsible for the failure. We then look at pipe segment pair $n_2^{C_1}$ and $n_3^{C_1}$. There are three parent nodes. Since one of them, t_2, has only one child node, t_3, which is marked "infertile", we determine that N_3 is also responsible for the failure. Applying the same process in C_2, we discover that N_2 and N_3, but not N_4, are the responsible pipes in C_2.

After we have examined all the pairs of pipe segments in C and determined the responsible pipes, we can generate annotations recording the failure conditions. Let $\{N_{i_1}, \ldots, N_{i_l}\}$ where $i_k \in [1, \ldots, p]$ and $l \leq p$ be the list of pipes that are identified as responsible pipes. To record the failure condition involving these pipes in C, for each of the responsible pipe, N_{i_k}, we create an annotation "if $[N_{i_j} : \theta_{i_j}, C, \phi_{i_j}]$ for $1 \leq j \leq p$ and $j \neq k$, then $[N_{i_k} : \theta_{i_k}, C, \phi_{i_k}]$" where θ_{i_j} and ϕ_{i_j} are cells preceding and succeeding C along the channel for pipe N_{i_j} (they can be terminals). The meaning of this annotation is exactly the same as that used in the two-dimensional case. Continue with the previous example, in C_1 we generate annotations "if $[N_2 : T_2^1, C_1, C_2]$ and $[N_3 : T_3^1, C_1, T_3^2]$ then $[N_1 : T_1^1, C_1, C_2]$" for N_1, "if $[N_1 : T_1^1, C_1, C_2]$ and $[N_3 : (T_3^1, C_1, T_3^2)]$ then $[N_2 : T_2^1, C_1, C_2]$" for N_2, and "if $[N_1 : T_1^1, C_1, C_2]$ and $[N_2 : T_2^1, C_1, C_2]$ then $[N_3 : T_3^1, C_1, T_3^2]$" for N_3. In C_2 we generate annotations "if $[N_2 : C_1, C_2, T_2^2]$ then $[N_1 : C_1, C_2, T_1^2]$" for N_1 and "if $[N_1 : C_1, C_2, T_1^2]$ then $[N_2 : C_1, C_2, T_2^2]$" for N_2.

The annotations generated in the first step are usually too strong. If we use these annotations when we search for alternative channel(s), we may lose solutions that are in fact feasible. This is because when we generate annotations in the first step, we only look at individual cells. But the failure in one cell may be caused by constraints in the adjacent cells. The second step of the failure discovery algorithm is to correct this problem by merging annotations.

In the second step, we consider each of the pipes that have more than one annotation attached to it, and we try to merge them as much as possible. This is done by putting all these annotations in a queue Q. At each iteration, we remove one annotation, ann_i, from Q and try to merge it with each of the remaining annotations in Q. If we are successful in merging ann_i with ann_j, we replace ann_j in Q by the merged annotation and put all annotations back in Q again. This process will terminate when Q becomes empty. Below we explain how to determine whether two annotations can be merged or not, and if so, how to construct the merged annotation.

Let us assume that pipe N_s has two annotations "if $[N_{i_k} : cl1_{i_k}]$ for $k = 1, \ldots, l_1$ then $[N_s : cl1_s]$" and "if $[N_{j_k} : cl2_{j_k}]$ for $k = 1, \ldots, l_2$ then $[N_s : cl2_s]$" where $cl1_i$ and $cl2_j$ are lists of cells. We say that these two annotations can be merged iff the following two conditions hold:

- $[N_s : cl1_s]$ can be merged with $[N_s : cl2_s]$;

- $\exists l \in [1, \ldots, l_1]$ and $\exists t \in [1, \ldots, l_2]$ such that $[N_{i_l} : cl1_{i_l}]$ can be merged with $[N_{j_t} : cl2_{j_t}]$.

$[N_g : cl_g]$ can be merged with $[N_h : cl_h]$ iff $N_g = N_h$ and the cell lists cl_g and cl_h are "connected", i.e. the first two cells or the last two cells of cl_g are the same as the first two cells or the last two cells of cl_h. Merging $[N_g : cl_g]$ and $[N_h : cl_h]$, we get $[N_g : cl]$, where cl is obtained by connecting cl_g and cl_h (at the point where they share the same cells).

If two annotations can be merged, we replace them with the merged annotation. The merged annotation is constructed as follows. The *then-part* of the merged annotation is the result of merging the *then-parts* of these two annotations. The *if-part*

of the merged annotation is the union of the *if-parts* of the two annotations, except that the components of the *if-parts* that can be merged are replaced by the merged components.

After we have completed merging all the annotations, we examine each of these annotations to see whether it can be made even weaker by replacing its *if-part* by a stronger condition. This is done by comparing each component of the *if-part* of this annotation to the *then-part* of all other annotations. Let $[N_g : cl_g]$ be a component of the *if-part* of this annotation and $[N_h : cl_h]$ be the *then-part* of another annotation, we say that $[N_g : cl_g]$ is weaker than $[N_h : cl_h]$ iff $N_h = N_g$ and cl_h contains cl_g. If this is the case, we replace $[N_g : cl_g]$ by $[N_h : cl_h]$.

The intuition behind this annotation merging process can be made clearer by continuing with the previous example as shown in Figure 4.33. Let us first consider pipe N_1. There are two annotations attached to it: "if $[N_2 : T_2^1, C_1, C_2]$ and $[N_3 : T_3^1, C_1, T_3^2]$ then $[N_1 : T_1^1, C_1, C_2]$" (derived in C_1); and "if $[N_2 : C_1, C_2, T_2^2]$ then $[N_1 : C_1, C_2, T_1^2]$" (derived in C_2). These two annotations can be merged because $[N_1 : T_1^1, C_1, C_2]$ and $[N_1 : C_1, C_2, T_1^2]$ can be merged and $[N_2 : T_2^1, C_1, C_2]$ and $[N_2 : C_1, C_2, T_2^2]$ can be merged. The merged annotation is "if $[N_2 : T_2^1, C_1, C_2, T_2^2]$ and $[N_3 : T_3^1, C_1, T_3^2]$ then $[N_1 : T_1^1, C_1, C_2, T_1^2]$". Similarly the two annotations attached to N_2 can be merged and we get "if $[N_1 : T_1^1, C_1, C_2, T_1^2]$ and $[N_3 : T_3^1, C_1, T_3^2]$ then $[N_2 : T_2^1, C_1, C_2, T_2^2]$". Although there is only one annotation attached to N_3, the conditional part of this annotation can be made stronger. The new annotation for N_3 is "if $[N_1 : T_1^1, C_1, C_2, T_1^2]$ and $[N_2 : T_2^1, C_1, C_2, T_2^2]$ then $[N_3 : T_3^1, C_1, T_3^2]$".

4.4 Implementation and Experimentation

The parallel routing algorithm (for the three-dimensional case) described above has been implemented. The parallel router is implemented in C and runs on a DecStation 3100. We have conducted many experiments with this parallel pipe router. We report four sets of such experiments below.

In the first set of experiments, there are 9 obstacles in a cubical workspace. The

terminals of the pipes are generated randomly. A terminal lies either on a surface of an obstacle or on the boundary of the workspace. We have generated and solved examples with up to 50 pipes. Some of the important statistics (calculated from many examples) are compiled in Table 4.1. Two of these examples are shown graphically in Figures 4.36 and 4.37. There are 20 pipes in the example shown in Figure 4.36 and the total running time for this example is 14.0 seconds. There are 40 pipes in the example shown in Figure 4.37 and the total running time for this example is 31.1 seconds.

Num of Pipes	Channel Generation	Refinement	Opt. Route Generation	Total
10	1.5	2.9	2.1	6.5
20	2.9	5.3	5.4	13.6
30	4.1	8.4	9.4	21.9
40	5.9	15.5	14.1	35.5
50	7.2	26.7	19.3	53.2

Table 4.1: Running time of the first set of experiments (in seconds)

In the second set of experiments, there are 27 obstacles equally distributed in a cubical workspace. The terminals of the pipes are generated randomly. A terminal lies either on a surface of an obstacle or on the boundary of the workspace. We have generated and solved examples with up to 50 pipes. Some of the important statistics are compiled in Table 4.2. Two of these examples are shown graphically in Figures 4.38 and 4.39. There are 30 pipes in the example shown in Figure 4.38 and the total running time for this example is 16.9 seconds. There are 40 pipes in the example shown in Figure 4.39 and the total running time for this example is 32.2 seconds. By comparing Table 4.1 and Table 4.2, we observe that the running time increases (as a function of the number of pipes) faster when the number of obstacles in the workspace is smaller. This is because when the number of obstacles in the workspace is smaller, the number of cells resulting from the decomposition is also smaller. Hence, it is more likely that many pipes are traversing the same cell, and therefore much more time is spent in channel refinement and optimal route generation even though less time is spent on initial channel generation.
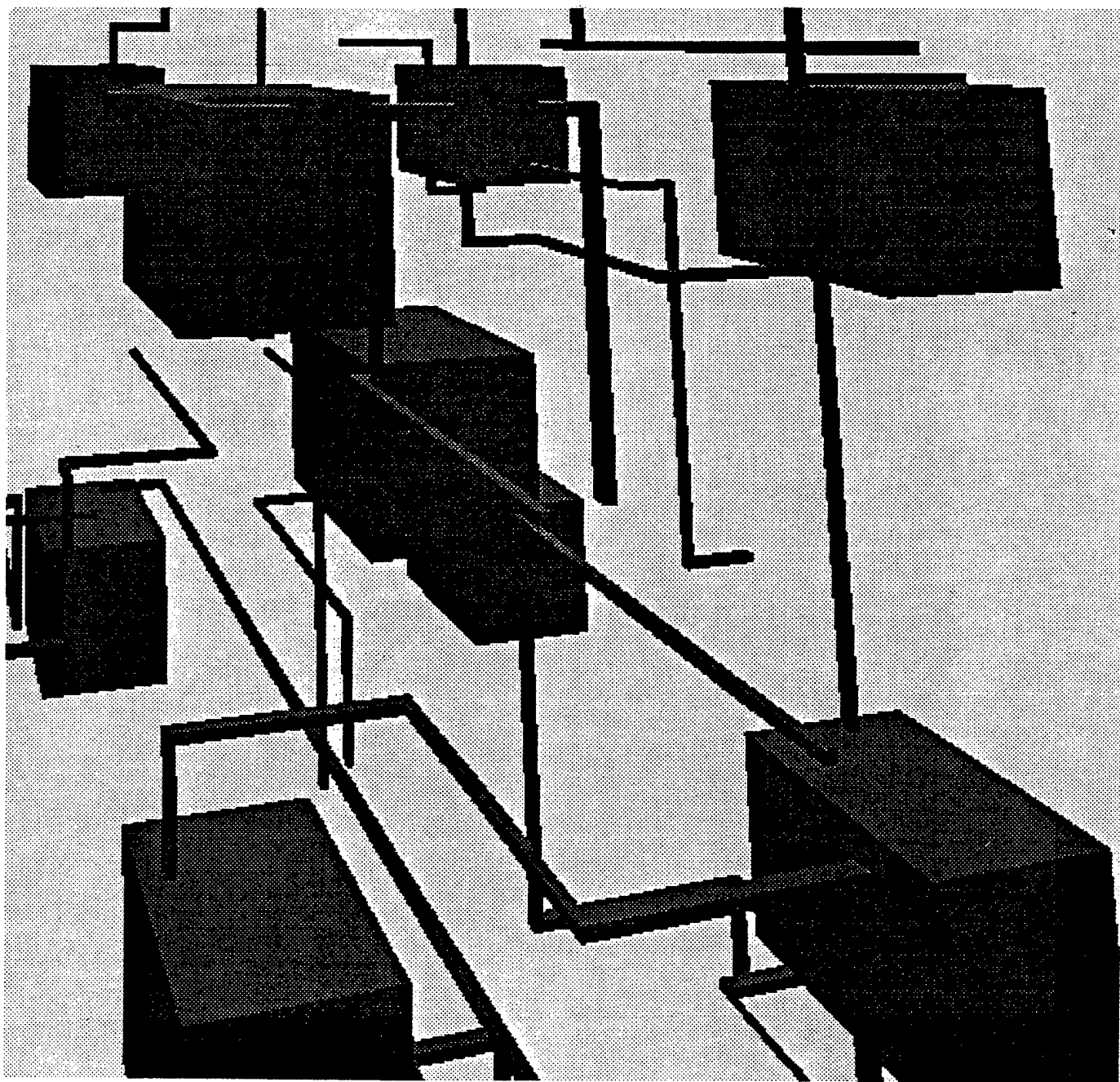
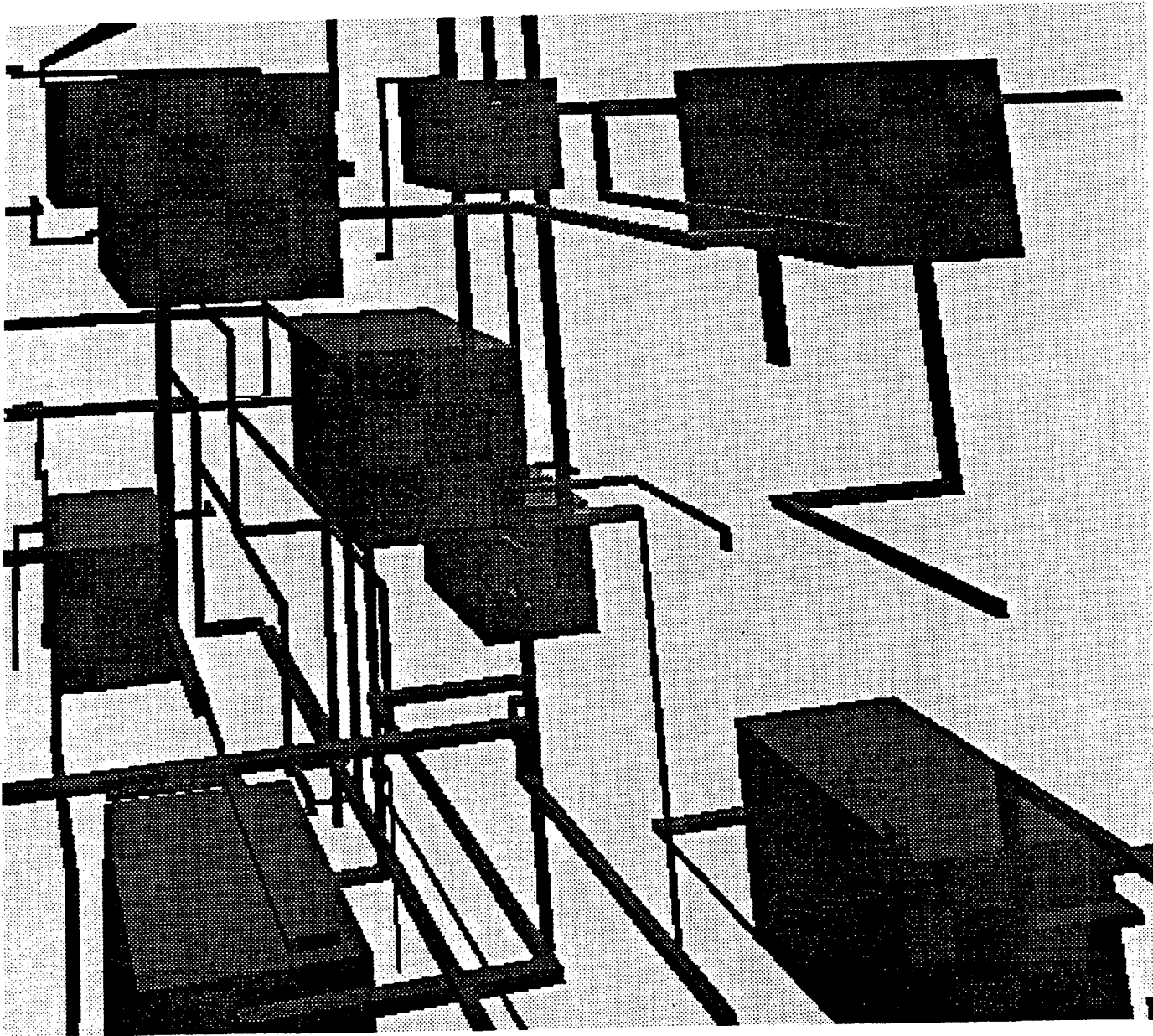Figure 4.36: An example from the first set of experiments.

Figure 4.37: Another example from the first set of experiments.

Num of Pipes	Channel Generation	Refinement	Opt. Route Generation	Total
10	2.0	1.9	1.0	4.9
20	3.9	4.3	2.7	10.9
30	5.7	6.4	5.6	17.7
40	8.5	12.5	10.2	31.2
50	10.4	20.7	16.5	47.6

Table 4.2: Running time of the second set of experiments (in seconds)

In the third set of experiments, there are 35 obstacles in a cubical workspace. Some of these obstacles form a frame structure that are common in many chemical and nuclear power plant designs. The other obstacles are inside this frame structure. The terminals of the pipes are generated randomly. A terminal lies either on a surface of an obstacle (not the frame structure) or on the boundary of the workspace. We have generated and solved examples with up to 50 pipes. The statistics are similar to that given in Table 4.2 above. Two of these examples are shown graphically in Figures 4.40 and 4.41. There are 30 pipes in the example shown in Figure 4.40 and the total running time for this example is 18.5 seconds. There are 40 pipes in the example shown in Figure 4.41 and the total running time for this example is 32.5 seconds.

In the last set of experiments, both the obstacles and the pipes are randomly generated (within a pre-specified cubical workspace). The terminals of the pipes are either on a surface of an obstacle or on the boundary of the workspace. We have generated and solved examples that consist of up to 15 randomly generated obstacles and up to 40 pipes. Some of these examples are shown graphically in Figures 4.42 and 4.43.

So far, our pipe router has solved pipe routing examples of up to 50 obstacles and 80 pipes. Compared to the pipe routers described in [Suwa et al, 1990] and [Narikawa et al, 1991] which have reported the routing of 10 pipes or less, our parallel router not only can handle many more pipes but also is much more efficient.
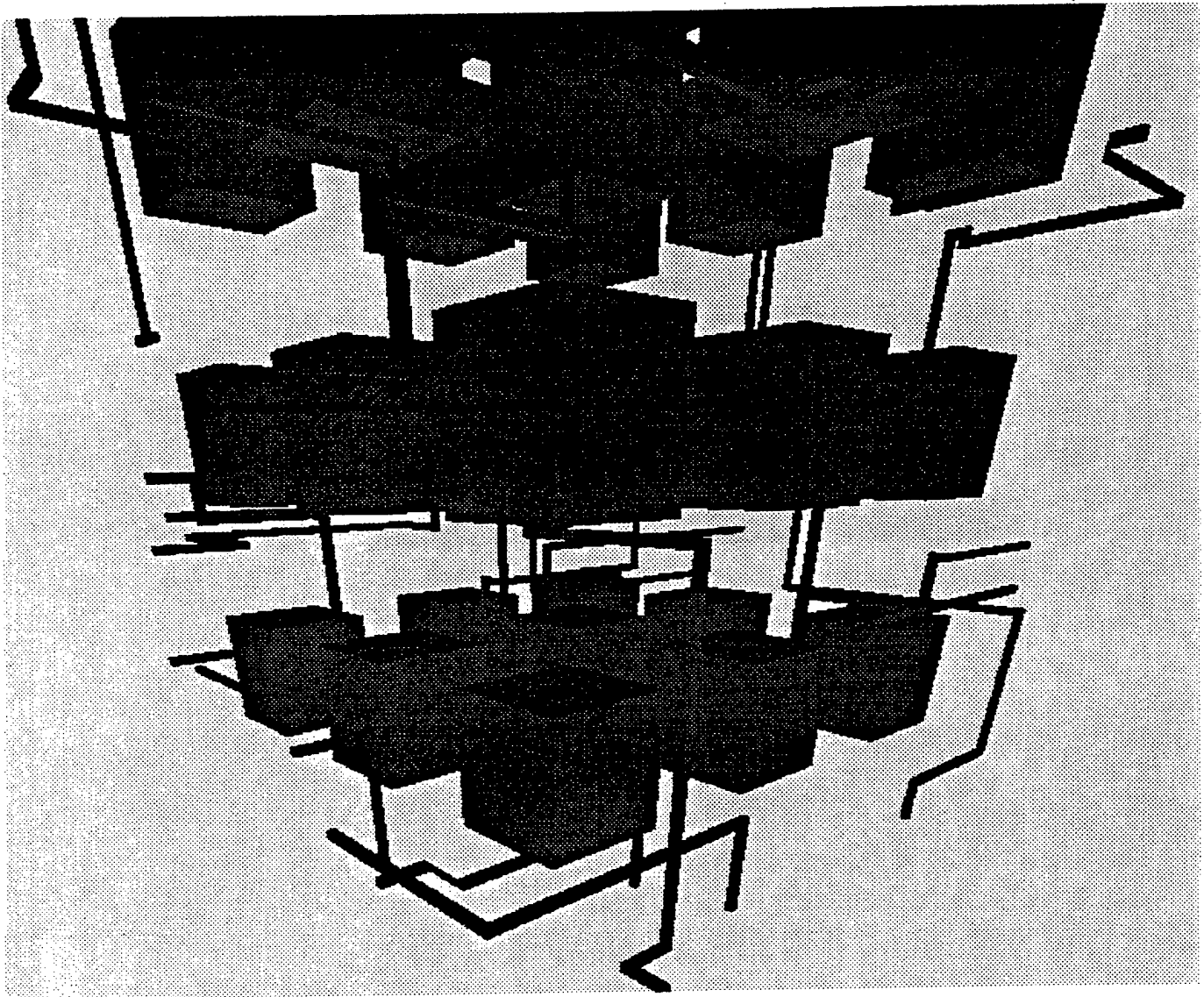
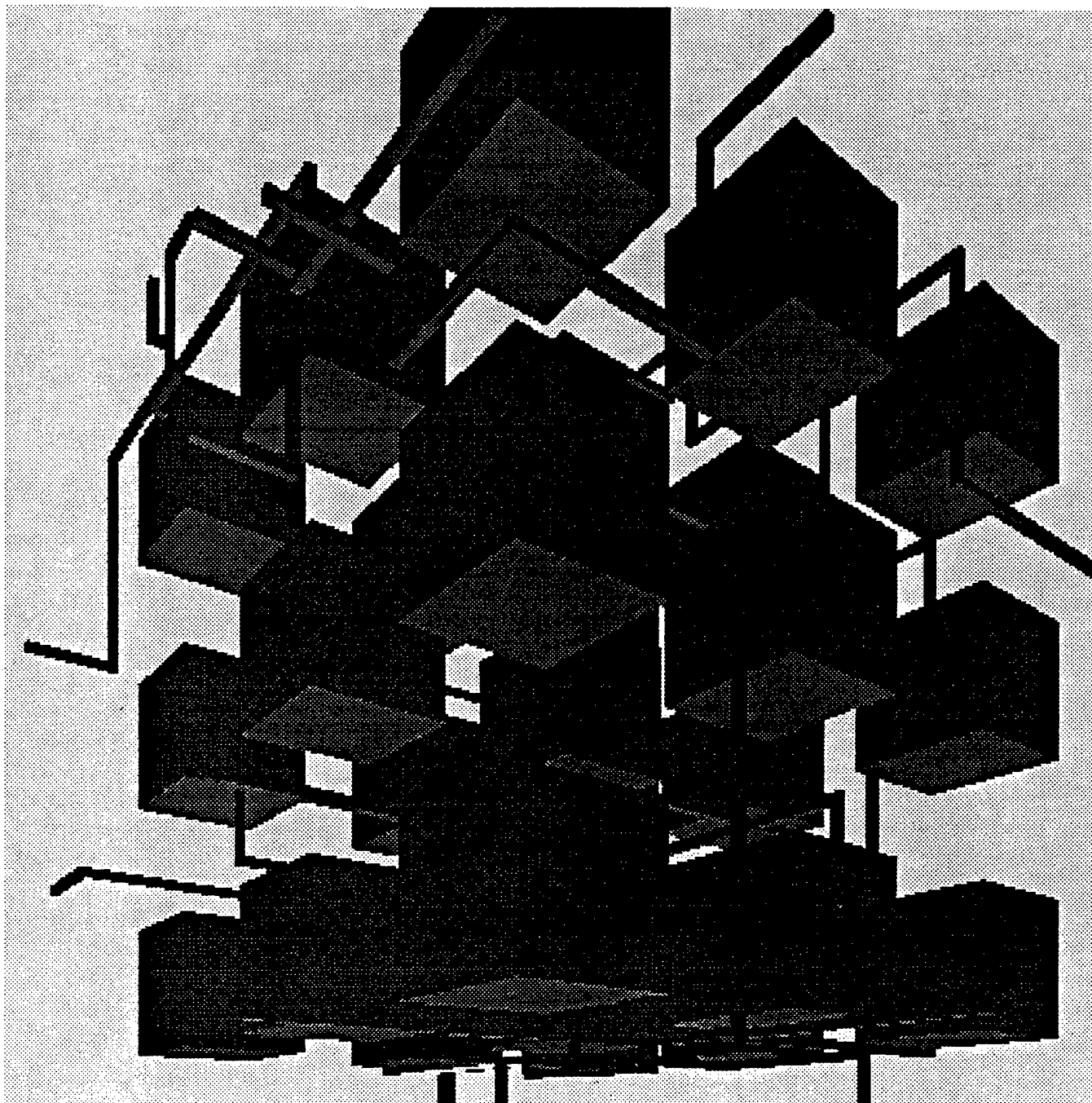Figure 4.38: An example from the second set of experiments.

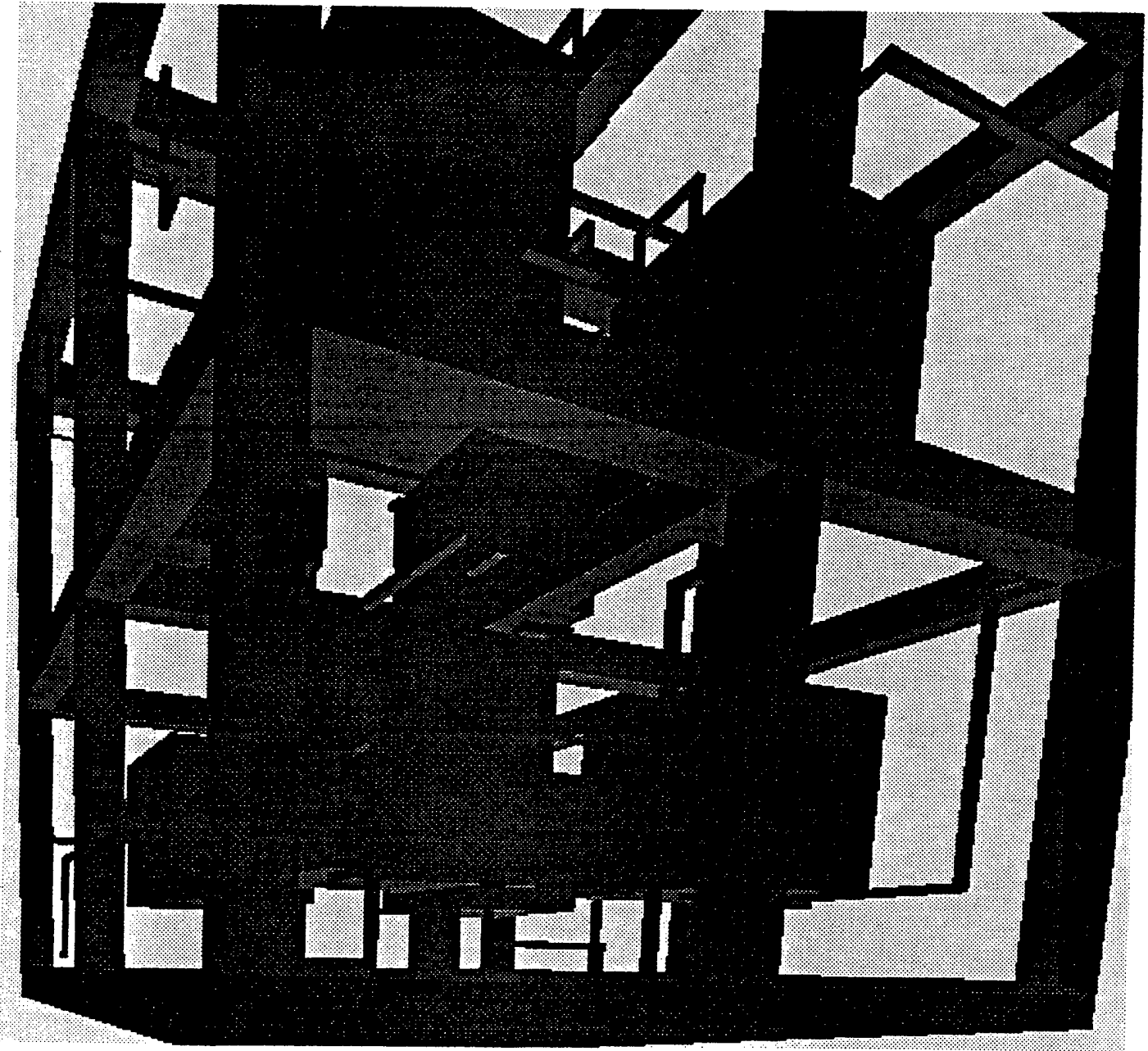Figure 4.39: Another example from the second set of experiments.

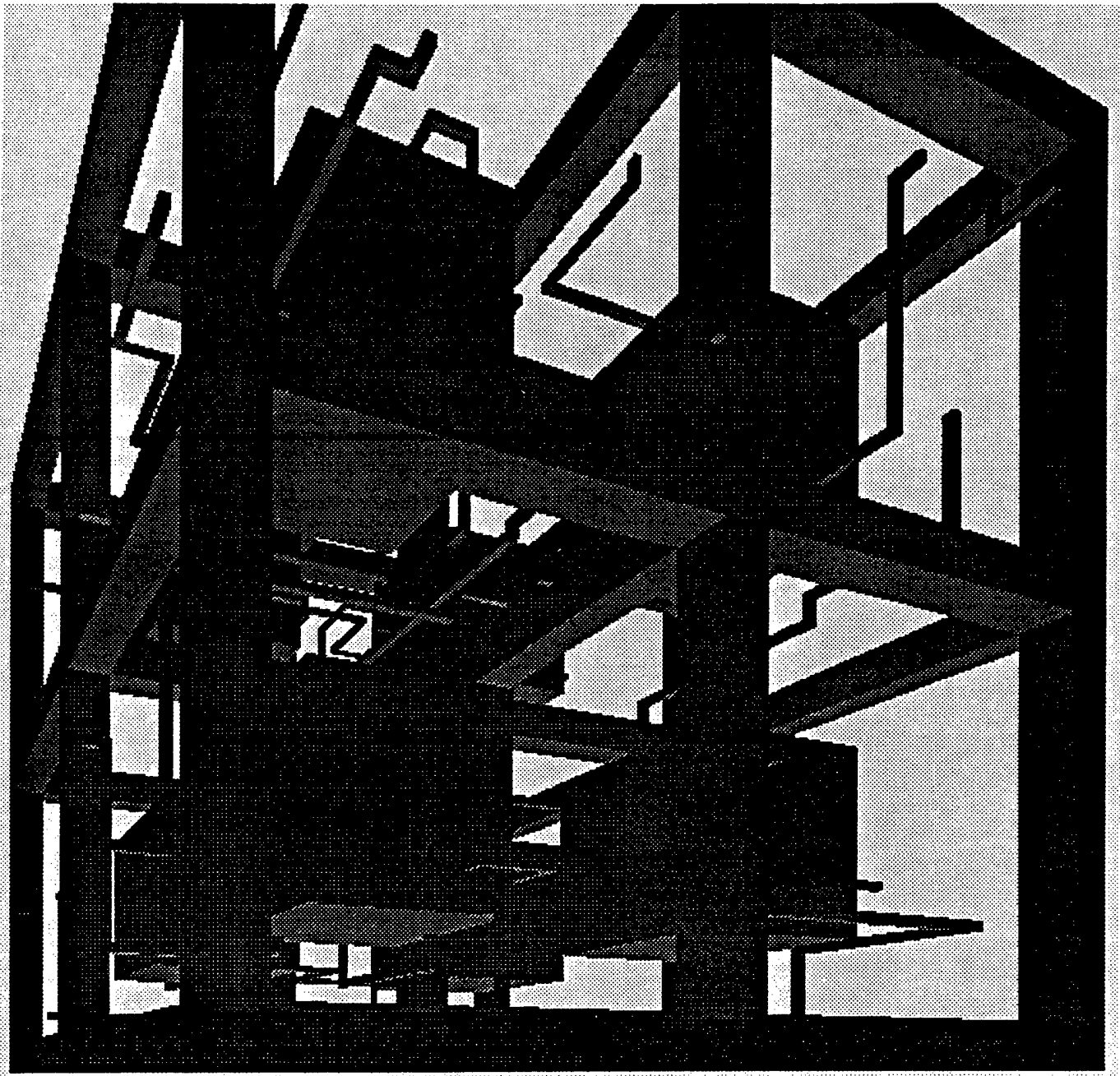Figure 4.40: An example from the third set of experiments.

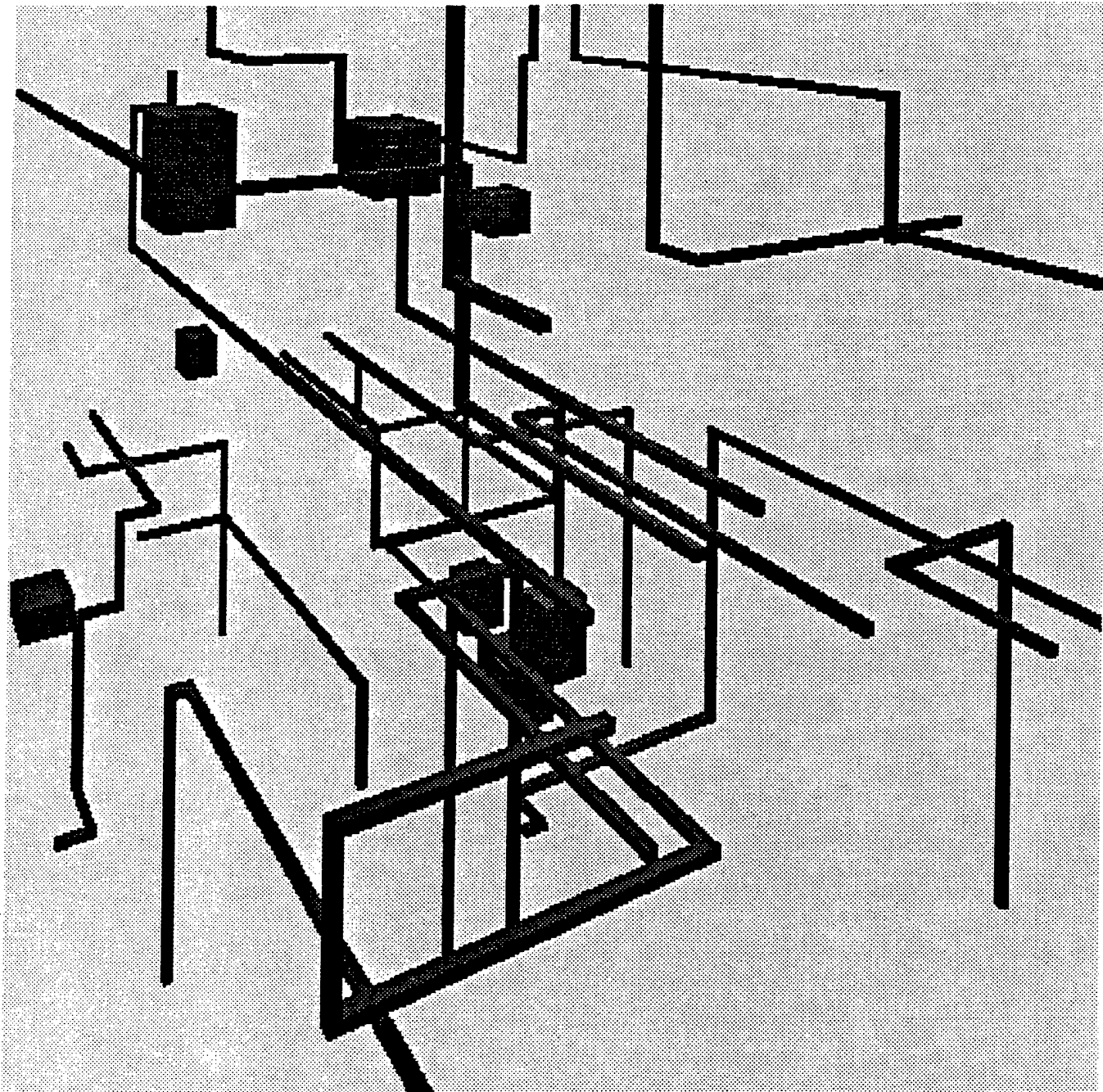Figure 4.41: Another example from the third set of experiments.

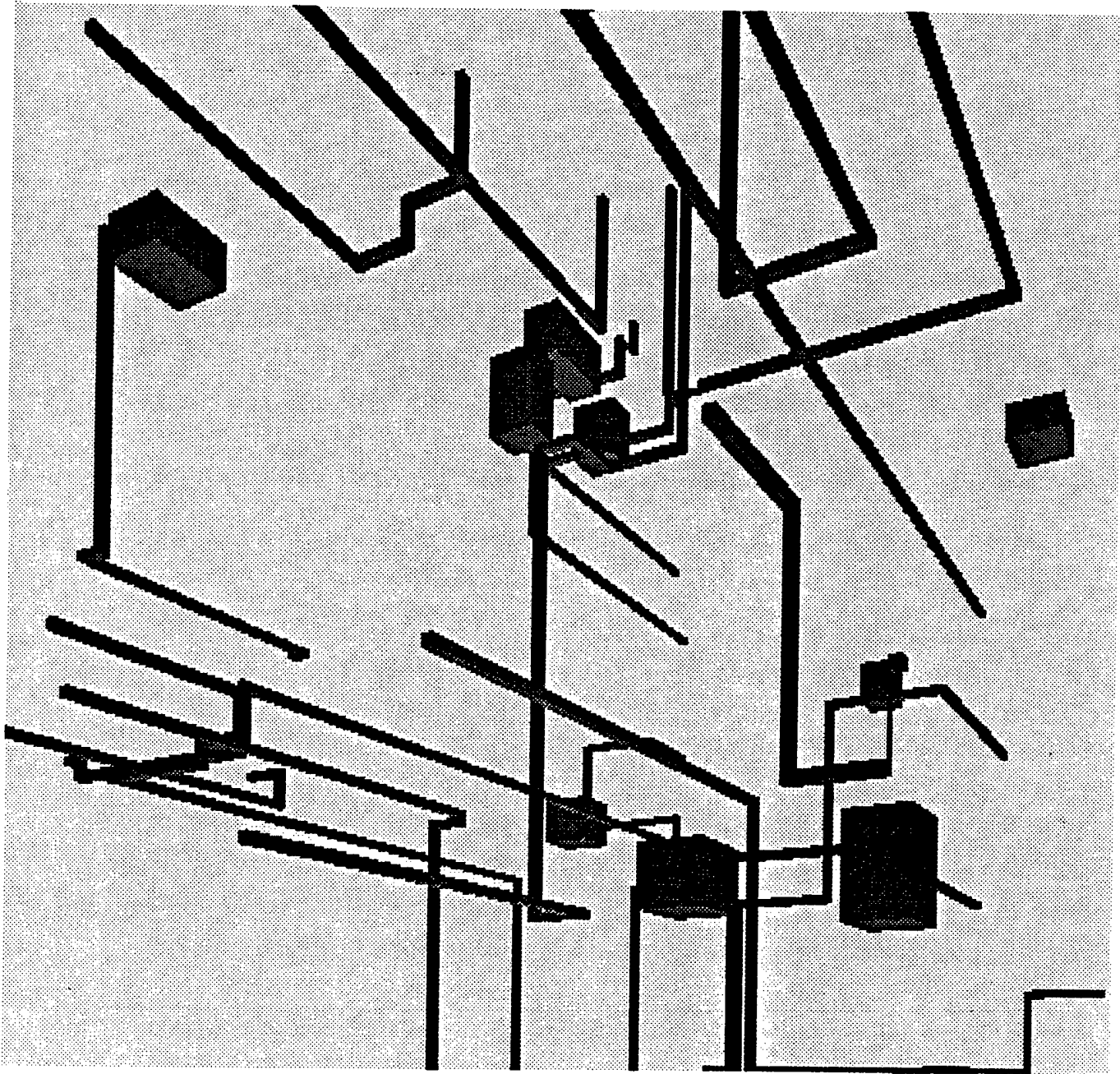Figure 4.42: An example from the fourth set of experiments.

Figure 4.43: Another example from the fourth set of experiments.

4.5 Limitations

One of the limitations of the parallel routing algorithm developed above is the assumption that all the pipes have the same radius. This assumption allows us to use a common configuration space for all pipes, and thus one connectivity graph represents the decomposition of this space. When two pipes have different radii, their configuration spaces are different. In this case, we must search their corresponding connectivity graphs to find channels for them. The cells along these two channels may no longer coincide but may overlap. However, the channel refinement algorithms developed in this chapter can be applied with minor modification, i.e. instead of applying it to a cell that is traversed by more than one pipe, we apply it to the non-empty intersection of several cells, each of which is traversed by one or more pipes.

Another limitation of the parallel routing algorithm is in the channel generation process. In the algorithm described above, the channel for each pipe is generated independently of the other pipes. Congestion may occur in a region if too many pipes are going through this region. This will cause failure in channel refinement and will result in backtracking. This simple channel generation algorithm can be modified so that it assesses how congested a cell is before including this cell in the channel for a pipe. By doing so we can increase the chance of success during channel refinement. But this modified channel generation algorithm is still short-sighted in the sense that a pipe considered later may have an unnecessarily long channel in order to avoid some congested regions, or may not have a channel at all. Hence the channel generation process can be further improved by allowing backtracking within the channel generation loop, i.e. if we found a channel for a pipe that is unacceptably long (or could not find one at all), we backtrack and change the channels of some other pipes that have caused the congestion. The selective backtracking algorithm developed for sequential routing can be applied here. An optimal channel generation algorithm based on *optimal spanning forest* in which channels are generated for all pipes in parallel has been proposed for VLSI global routing [Cong, 1990]. This parallel channel generation algorithm can be modified for the pipe routing application as well.

4.6 Summary

In this chapter we continued to explore the problem decomposition approach in the context of pipe routing. We addressed the problem of how to reduce the need for backtracking when the interactions among the subproblems (pipes) are stronger. We have developed a parallel routing method based the lesser-commitment approach. The parallel router considers the interactions among the pipes before any route is generated in order to eliminate harmful interactions among the pipes. It consists of five modules:

- Channel generation

- Constraint Derivation

- Channel refinement

- Route generation

- Backtracking

We have developed this method for the two-dimensional case and have extended it to the three-dimensional case. The main algorithms developed in this chapter are:

- Constraint propagation algorithms for constraint derivation and channel refinement.

- Failure condition recording algorithms for recording channel refinement failure.

- Selective backtracking algorithms for generating alternative relative constraints and for generating alternative channels.

Chapter 5

Conclusion

5.1   Summary of Contribution

We addressed the problem of developing path planning algorithms that are both efficient and well-behaved. We proposed a novel approach in which we solve a path planning problem by finding and solving an appropriate abstraction of the original problem. We argued that in order for this approach to be efficient, a tighter integration of geometric reasoning and search is essential. This thesis developed evidence, both theorectical and experimental, to support this argument. In particular, we investigated in-depth two approaches for generating problem abstractions: the constraint approximation approach (in the context of robot motion planning); and the problem decomposition approach (in the context of pipe routing). For each of these two approaches, we developed algorithms that tightly integrate geometric reasoning with search and we addressed many issues raised by this integration.

For the constraint approximation approach, we have developed a new method for approximate cell decomposition based on constraint reformulation and an efficient hierarchical search algorithm with error discovery and recording mechanisms.

- The new approximate cell decomposition method, by using information of the constraints to guide the decomposition, results in a much more efficient decomposition of the space (in terms of time, the resulted number of cells, and the

volume of MIXED regions generated) compared to the traditional "divide-and-label" methods.

- The new hierarchical search algorithm, having the ability to discover and remember failure conditions, prevents the planner from making the same mistakes again. Except in some pathological cases, this new algorithm is better than the naive hierarchical search algorithm (described in Subsection 2.3.1) used in previous hierarchical path planners.

Both the approximate cell decomposition algorithm and the hierarchical search algorithm have been implemented in a path planner intended to be part of a mobile robot navigation system for a two-dimensional polygonal robot allowed to translate and rotate.

For the problem decomposition approach, we investigated two ways of dealing with subproblem interaction: sequential routing with selective backtracking; and parallel routing based on constraint propagation techniques.

- In sequential routing, we have developed a selective backtracking algorithm with a net-protection scheme. The net-protection scheme prevents the backtracking algorithm from looping and allows it to detect and recover from a backtracking error. This backtracking algorithm is an improvement over chronological backtracking and selective backtracking without the net-protection scheme.

- In parallel routing, we developed geometric constraint propagation techniques that enable early consideration of the interaction among subproblems before specific routes are generated. The early consideration of subproblem interaction reduces the need for backtracking. It provides a more efficient solution to pipe routing problems that involve stronger pipe interactions.

Both the sequential routing and the parallel routing approaches have been implemented. The parallel router has generated complex pipe routes involving several tens of pipes whose nozzles are randomly distributed in a three-dimensional space.

To our knowledge, this thesis is the first systematic investigation of the importance of integrating geometric algorithms with search algorithms for solving complex geometric reasoning problems.

5.2 Direction of Future Work

Although we obtained strong supporting evidence for our argument that tighter integration between geometric algorithms and search algorithms can improve the efficiency of solving path planning problems, there are several issues that require further investigation . As we have pointed out earlier, there are limitations to both the general approach (discussed in Section 1.5) and to our implementation of the approach (discussed in Sections 2.5, 3.8, and 4.5). It is important to have a better understanding of these limitations in order to determine under what conditions the approach and our implementation would not work well, and to provide remedies for these situations. Below we discuss several specific points that we believe deserve special attention.

In our implementation of the approximate cell decomposition method, we have chosen rectangloids as the primitive for approximating obstacles and for decomposing the space. But as the space becomes more cluttered, a better approximation of the obstacles is necessary in order to find a solution (or the absence of one). In this case, rectangloids may not be the best choice (Section 2.5). What other more complex geometric shapes can we use to better approximate the obstacles? How do we decide when to use these more complex shapes? If we allow different shapes as primitives for decomposing the space, how do we maintain the hierarchy?

In our implementation of the constraint approximation approach for robot motion planning, we only approximate in the configuration space. This has the advantage that when we discover that an approximation is too inaccurate to admit a solution, we have direct information to guide the refinement of the approximation. But for many practical problems, a large portion of the workspace is irrelevant. In these cases, the cost of computing and processing the irrelevant C-obstacles may become a major burden on the cost for solving the problem. In Section 2.5, we suggested the

use of a "scope". But questions remain as to how to choose the initial scope and how to enlarge the scope once we cannot find a solution within the current scope.

In our implementation of the sequential routing approach, we sacrifice completeness to gain efficiency by restricting the backtracking space. We provided an example in Section 3.8 that cannot be solved by our backtracking algorithm. How large is this class of problems in the two-dimensional workspace? and in the three-dimensional workspace? We can make the backtracking algorithm more complete by enlarging the backtracking space. What is the trade-off in efficiency? How can these backtracking algorithms work together, i.e., we first use a more efficient backtracking algorithm but when we fail to find a solution we switch to a more complete but less efficient algorithm?

In our implementation of the parallel routing approach, we initially generate the channels more or less blindly. As we pointed out in Section 4.5, some heuristics can be used to guide the generation of the channels to help increase the success rate of the subsequent channel refinement. What is a good set of heuristics? Furthermore, the channels are generated sequentially. Can we generate the channels in parallel, for example, by adapting the optimal spanning forest method proposed by [Cong, 1990] for VLSI global routing?

Throughout this thesis, we assumed that the use of annotations can improve the efficiency of search by preventing the same mistakes from happening again. Experimental results support this assumption. However there is a cost associated with the generation and the use of the annotations. One can imagine that in some pathological cases, the cost of generating and using annotations outweighs the benefits we gain from using them. Can we characterize these cases for motion planning problems? and for pipe routing problems?

Despite these limitations, we have shown that the approach and the techniques described in this thesis allow us to develop efficient and reliable algorithms for solving path planning problems. We believe that both the approach and the techniques are general and can be applied to many other problems that involve both geometric (or

numeric) computation and symbolic reasoning.

Appendix A

Related Work on Routing

In this thesis, the problem of pipe routing serves as the context for exploring the problem decomposition approach. Although it is not the ultimate goal of this thesis to solve the pipe routing problem, we believe that a brief survey of some previous work on pipe routing and VLSI routing can provide a different prospective of our research.

A.1 Pipe Routing

The problem of pipe routing has been addressed before in the development of several pipe routing systems. These existing pipe routing systems can more or less be classified into the two approaches that were described in this thesis: sequential pipe routing and parallel pipe routing.

Sequential Pipe Routing [Mitsuta et al, 1986] describes a pipe routing system developed in Hitachi that is based on the sequential pipe routing approach. Its basic routing procedure is similar to that described in Chapter 3 of this thesis. The difference lies in that instead of explicitly growing the obstacles and searching in the configuration space, it searches in the work space and checks for collision between pipes and obstacles during search. A similar pipe design system was also developed in Bechtel [Bechtel, 1986]. [Zhu and Freeman, 1988] and [Wangdahl et al, 1974] describe

two other pipe routing systems based on the sequential approach. The common drawback of these routing systems is that they do not consider backtracking (or only at a superficial level). Since backtracking is required when a system cannot find a route for a pipe due to some of the previously routed pipes, these approaches cannot handle routing problems that have stronger pipe interactions.

Parallel Pipe Routing We are aware of two routing systems ([Suwa et al, 1990] and [Narikawa et al, 1991]) that are more in the spirit of the parallel routing approach described in Chapter 4 of this thesis. [Suwa et al, 1990] describe a pipe design system used in IHI that first generates a set of *piping passages*. It searches the graph of piping passages to find an optimal sequence of piping passage for each pipe (no specific route is generated within the piping passages). This first step is very similar to the channel generation step of our parallel routing algorithm. In the second step of the IHI pipe routing system, the locations of the pipes within each piping passage is set by searching the space of possible pipe arrangements within the piping passage. A similar two-phase pipe routing system developed in Toshiba is described in [Narikawa et al, 1991]. But after a sequence of piping passages (called rooms in this system) is assigned to a pipe, the Toshiba pipe router assigns detailed route to each pipe independent of the other pipes. Hence two pipe routes may overlap. In the second phase, the overlapping pipes are modified locally, again by searching the space of possible pipe arrangements within a room.

The main drawback common to both of these two pipe design systems is that there is no backtracking from the pipe arrangement phase to the initial routing phase. They assume that pipes can always be arranged locally to obtain a solution. Another problem with these two routing systems is that is not clear from their descriptions how the systems, after arranging the pipes in a region, propagate the constraints to the adjacent regions. This is the problem addressed by the constraint derivation process in our parallel routing algorithm.

A.2 VLSI Routing

Although there is certainly similarity between VLSI routing and pipe routing, the constraints to be satisfied by the routes in the two problems are quite different. VLSI routing is strictly layered and further constraints are imposed in terms of wiring models. These constraints drastically reduce the search space for the VLSI routing problems which makes certain combinatoric algorithms feasible for solving them. Although these constraints are practical or even desirable for VLSI routing, they are too constraining for many pipe routing problems. However, some of the ideas and techniques developed in VLSI routing are applicable to pipe routing.

There are two general approaches to VLSI routing which we will call one-phase routing and two-phase routing[1]. In the one-phase routing approach, a route is generated for a net at a time in a sequential fashion. In the two-phase routing approach, the routing of the nets is carried out in two steps: global routing and detailed routing. In the global routing step, a routing region is assigned to each net. In the detailed routing step, a route is generated for every net within its routing region. The one-phase routing is similar to the sequential pipe routing approach described in Chapter 3, while the two-phase routing is more in the spirit of the parallel routing approach described in Chapter 4. We provide a brief survey of these two approaches below.

One-phase Routing The two most popular algorithms for this approach are the Lee-Moore routing algorithm (which is also referred to as the Maze routing algorithm) [Lee, 1961] and the Hightower routing algorithm (which is also called line-routing algorithm) [Hightower, 1969]. The Lee-Moore algorithm employs breadth-first search while the Hightower algorithm uses bi-direction search. Both algorithms route one net at a time and treat previously routed nets as obstacles.

The main problem with these two algorithms is that the nets that are to be routed later may be obstructed by nets that have been routed earlier. If this happens, some of the previously routed nets have to be ripped up and re-routed. The main difficulties in the rip up and re-route process are to identify the appropriate net(s) to rip up and

---

[1]This terminology is not common in VLSI routing literature.

to re-route them correctly. Care must be taken to prevent the rip up and re-route process from falling into a loop. Several techniques have been developed for ripping up and re-routing nets [Shin and Sangiovanni-Vincentelli, 1986] and [Poirier, 1987]. But they have not fully addressed the two issues raised above which are addressed by our selective backtracking algorithm with net-protection scheme.

Two-phase Routing Even with intelligent rip up and re-route, the one-phase routing approach is not efficient for large problems. Most of the practical routers used today are based on the two-phase approach. In the first phase, which is referred to as global routing, the available routing area is decomposed into regions of simple shapes (these regions are often called channels in the VLSI routing literature [2]) and for each net, a general routing area (in terms of a list of simple regions) is assigned to it. In the second phase, which is called detailed routing or channel routing, a detailed route is chosen within a region for every net that traverses that region. We describe these two steps below.

After the routing area has been decomposed, the available routing area can be represented as a graph. A node corresponds to a routing region and there is an edge between two nodes iff the regions corresponding to these two nodes are adjacent. Any standard search technique can be used to search this graph for a path for every net. [Cong, 1990] describes a parallel algorithm for global routing based on optimal spanning forest.

Once we have assigned a routing area for every net, the routing problem is decomposed into a set of smaller subproblems. Each subproblem consists of finding route(s) within each of those regions that are traversed by one or more pipes. In most routers, these regions are rectangular. These subproblems are called channel routing problems if the nets traverse the rectangular regions on two parallel edges as shown in Figure A.1a. They are called switch-box routing if the nets are allowed to traverse the rectangular regions on all four edges, as shown in Figure A.1b. The switch-box routing problem is significantly more complex than the channel routing

---

[2]Channels here refer to the simple regions which are called cells in this thesis.

problem and is less studied. On the other hand, many of the issues are the same between channel routing and switch-box routing. Hence, we will only provide a survey of channel routing methods below.
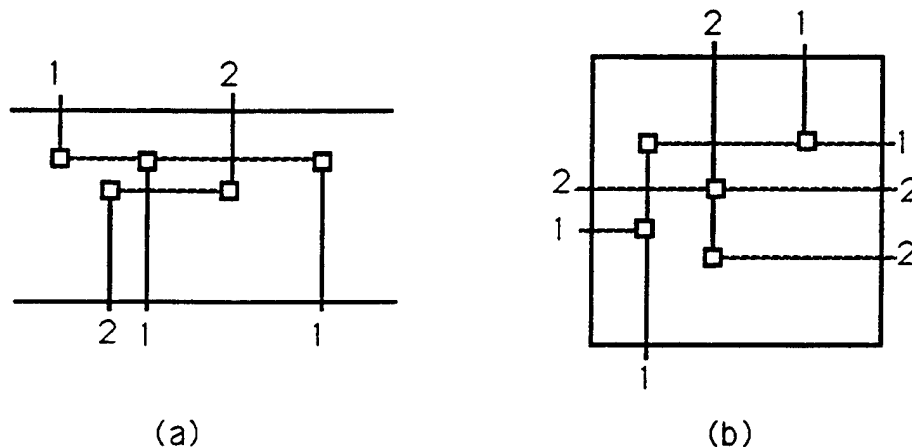

(a)   (b)

Figure A.1: Examples of channel routing and switch-box routing problems

There has been a large amount of research on channel routing and many algorithms have been proposed. A general paradigm assumed by most channel routing algorithms is that for each net in a channel, there is a set of possible routes for this net. We call this set *route set* of the net. Most channel routing algorithms essentially search in a space that is the cross product of the route sets of all the nets in the channel. The route set for a net can be very large. Therefore most algorithms restrict the route set for a net by imposing constraints on what kind of routes are feasible. These constraints are called *wiring models*. One of the most commonly used wiring model is called *the Manhattan model* which imposes that the horizontal lines of a route must lie in one layer and the vertical lines of the route lie in the other. Further restriction can be made to the Manhattan model which imposes the constraint that each route can only have one horizontal wire segment. When the routing sets of the nets are restricted to a manageable size, an exhaustive search of the space is feasible [Kernighan et al, 1973]. Of course, when you restrict the route sets, you lose solutions. For example, the routing problem shown in Figure A.2 can not be solved with the one track per net constraint.
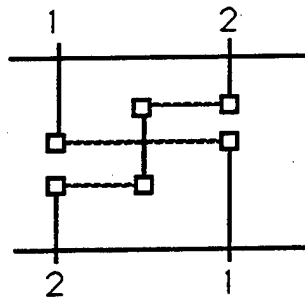
Figure A.2: An example that can not be solved with one track per net constraint

When the route sets are too large for exhaustive search, enhancement of the search algorithms can be made in one of several ways: 1) refining the route sets of the nets by considering the interactions among the nets; 2) using heuristics to guide the search; 3) decomposing the problems into simpler subproblems. The route sets can be refined by considering the *vertical constraints* among the nets. There is a vertical constraint from net_2 to net_1 iff a top terminal of net_2 lies in the same column as a bottom terminal of net_1 as shown in Figure A.3. When there is a vertical constraint from net_2 to net_1, we can refine these route sets of them by considering the fact that the route of net_2 must lie above that of net_1 in the constraining column. There are several algorithms that exploit this idea in various ways [Hashimoto and Stevens, 1971] [Deutsch, 1976] [Yoshimura and Kuh, 1982]. [Rivest and Fiduccia, 1982] uses a set of "greedy" heuristics to guide the search. [Burstein and Pelavin, 1983] takes a "divide-and-conquer" approach that recursively divides the problems into a set of easier subproblems.
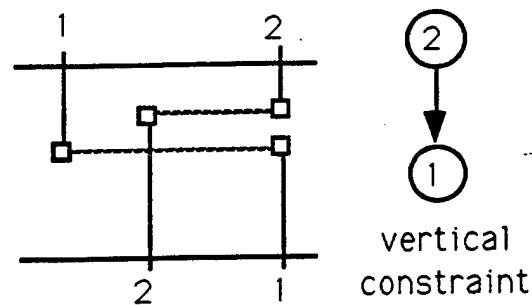
Figure A.3: Vertical constraint

In the two-phase algorithms described above, the two phases of the routing process are separated. In the global routing phase, when the route of a net traverses a channel, the "pseudo-pin" locations of this net on the boundary of the channel are determined. The choice of these pseudo-pin locations during global routing may be poor because of the lack of information about other nets. These poor choices may lead to poor channel routing results even though the channel router itself is good. [Tzeng, 1988] proposes a two-phase routing algorithm that integrates global and detailed routing. In this algorithm, it is possible for the detailed routing phase to backtrack to the global routing phase in case no good detailed routing solutions can be found.

Reference

Aho, A.V., Hopcroft, J.E. and Ullman, J.D. (1983) *Data Structures and Algorithms*, Addison-Wesley, Reading, MA.

Avnaim, F. and Boissonnat, J.D. (1988) *Polygon Placement Under Translation and Rotation*, Technical Report No. 889, INRIA, Sophia-Antipolis, France.

Ayala, D., Brunet, P., Juan, R. and Navazo, I. (1985) "Object Representation by Means of Nonminimal Division Quadtrees and Octrees," *ACM Transactions on Graphics*, 4(1), 41-59.

Barraquand. J. and Latombe, J.C. (1989) "On Non-Holonomic Mobile Robots and Optimal Maneuvering," *Revue d'Intelligence Artificielle*, 3(2), Hermes, Paris, 77-103.

Barraquand, J. and Latombe, J.C. (1991) "Robot Motion Planning: A Distributed Representation Approach," *International Journal of Robotics Research*, MIT Press, 10(6).

Bechtel (1986) "Expanding CAD Applications on Petroleum Projects," *Bechtel CAE Bulletin*.

Brooks, R.A. and Lozano-Pérez, T. (1985) "A Subdivision Algorithm in Configuration Space for Findpath with Rotation," *IEEE Transactions on Systems, Man and Cybernetics*, SMC-15(2), 224-233.

Brost, R.C. (1989) "Computing Metric and Topological Properties of Configuration-Space Obstacles," *IEEE International Conference on Robotics and Automation*, Scottsdale, Arizona.

Burstein, M. and Pelavin, R. (1983) "Hierarchical wire routing," *IEEE Trans. CAD of ICs and Systems*, CAD-2(4) 223-234.

Caloud, P., Choi, W.Y., Latombe, J.C., Le Pape, C. and Yim, M. (1990) "Putting many robots in the same environment," *IEEE International Workshop on Intelligent Robot and Systems (IROS*, Tsuchiura, Japan, 67-72.

Canny, J.F. and Donald, B.R. (1988) "Simplified Voronoi Diagrams," *Discrete and Computational Geometry*, Springer-Verlag, 3, 219-236.

Chambon, R. et al. (1987) "An Expert System for Objects Placing in Three-Dimensional Space," *KBES in Engineering: Planning and Design*, 447-459

Choi, W., Zhu, D.J. and Latombe, J.C. (1989) "Contingency-Tolerant Motion Planning and Control," *IEEE/RSJ International Workshop on Intelligent Robots and Systems (IROS)*, Tsukuba, Japan.

Choi, W.Y. and Latombe, J.C. (1991) "A Reactive Architecture for Planning and Executing Robot Motions with Incomplete Knowledge," *IEEE/RSJ International Workshop on Intelligent Robots and Systems (IROS)*, Osaka, Japan.

Christensen, J. (1990) "A hierarchical planner that generates its own abstraction hierarchies," *Proceedings of Eighth National Conference on Artificial Intelligence*, 1004-1009.

Clow, G.W. (1984) "A global routing algorithm for general cells," *Proceedings of 21st Design Automation Conference*, 45-51

Clowes, M.B. (1971) "On seeing things," *Artificial Intelligence* 2, 79-116.

Cong, J. (1990) *Routing Algorithms in the Physical Design of VLSI Circuits*, Ph.D. thesis, Department of Computer Science, University of Illinois at Urbana-Champaign.

Deutsch, D.N. (1976) "A Dogleg Channel Router," *Proceedings of 13th Design Automation Conference*, 425-433.

Donald, B.R. (1983) "The Mover's Problem in Automated Structural Design," *Proceedings of the Harvard Computer Graphics Conference*, Cambridge, MA.

Donald, B.R. (1984) *Motion Planning With Six Degrees of Freedom*, Report No. AI-TR-791, Artificial Intelligence Laboratory, MIT, Cambridge, MA.

Donald B.R. and Pai, D.K. (1989) *On the Motion of Compliantly-Connected Rigid Bodies in Contact, Part II: A system for Analyzing Design for Assembly*, TR 89-1048, Department of Computer Science, Cornell University, Itheca, NY.

Doyle, J. (1979) "A truth maintenance system," *Artificial Intelligence*, 12:231-272

Erdmann, M. and Lozano-Pérez, T. (1986) *On Multiple Moving Objects*, AI Memo No 883, Artificial Intelligence Laboratory, MIT, Cambridge, MA.

Faverjon, B. (1984) "Obstacle Avoidance Using an Octree in the Configuration Space of a Manipulator," *Proceedings of the IEEE International Conference on Robotics and Automation*, Atlanta, 504-512.

Faverjon, B. (1986) "Object Level Programming of Industrial Robots," *Proceedings of the IEEE International Conference on Robotics and Automation*, San Francisco, 1406-1412.

Faverjon, B. and Tournassoud, P. (1989) A Practical Approach to Motion Planning for Manipulators with Many Degrees of Freedom. *Preprints of the Fifth International Symposium of Robotics Research*, Tokyo, Japan, 65-73.

Fortune, S. and Wilfong, G. (1988) "Planning Constrained Motion," *Proceedings of the Fourth ACM Symposium on Computation Geometry*, 445-459.

Gouzènes, L. (1984) "Strategies for Solving Collision-Free Trajectories Problems for Mobile and Manipulator Robots," *International Journal of Robotics Research*, 3(4), 51-65.

Gunn, D.J. and Al-Asadi, H. D. (1987) "Computer-aided Layout of Chemical Plant: a Computational Method and Case Study," *Computer Aided Design*, 19 (3), 131-140.

Haralick, A.K. and Shaprio, L. (1979) "The consistent labeling problem: Part 1," *IEEE Trans. Pattern Anal. Machine Intell*, PAMI-1, 173-184.

Hasan, H. and Liu, C.L. (1986) "A Force-Directed Global Router," *Proceedings of the*

Stanford Conference on VLSI Design, 135-150.

Hashimoto, A and Stevens, J. (1971) "Wire Routing by Optimizing Channel Assignment within Large Apertures," *Proceedings of 8th Design Automation Workshop*, 155-169.

Hightower, D.W. (1969) "A solution to line-routing problem on the continuous plane," *Proceedings of the 6th Design Automation Workshop*, 1-24.

Hopcroft, J.E., Schwartz, J.T. and Sharir, M. (1984) "On the Complexity of Motion Planning for Multiple Independent Objects: PSPACE-Hardness of the 'Warehouse-man's Problem'," *International Journal of Robotics Research*, 3(4), 76-88.

Huffman, D.A. (1971) "Impossible objects as nonsense sentences," R. Meltzer and D. Michie editor *Machine Intelligence*, 6, Elsevier, New York, 295-323.

Jacobs, P. and Canny, J. (1989) "Planning Smooth Paths for Mobile Robots," *Proceedings of the International IEEE Conference on Robotics and Automation*, Scottsdale, AZ, 2-7.

Kambhampati, S. and Davis, L.S. (1986) "Multiresolution Path Planning for Mobile Robots," *IEEE Journal of Robotics and Automation*, RA-2 (3), 135-145.

Kant, K. and Zucker, S.W. (1986a) *Planning Smooth Collision-Free Trajectories: Path, Velocity and Splines in Free Space*, Technical Report, Computer Vision and Robotics Laboratory, McGill University, Montréal.

Kant, K. and Zucker, S.W. (1986b) "Toward Efficient Trajectory Planning: Path Velocity Decomposition," *International Journal of Robotics Research*, 5, 72-89.

Kernighan, B.W. et al (1973) "An optimal channel routing algorithm for polycell layouts of integrated circuits," *Proceedings of the Tenth Design Automation Workshop*, IEEE Computer Society, 50-59.

Khatib, O. (1986) "Real-Time Obstacle Avoidance for Manipulators and Mobile Robots," *International Journal of Robotics Research*, 5(1), 90-98.

Khosla, P. and Volpe, R. (1988) "Superquadric Artificial Potentials for Obstacle

Avoidance and Approach," *IEEE International Conference of Robotics and Automation*, Philadelphia, PA, 1178-1784.

Knoblock, C.A. (1990) "A theory of abstraction for hierarchical planning," In D. Paul Benjamin, editor, *Change of Representation and Inductive Bias*, Kluwer, Boston, 81-104,

Kobayashi, Y. et al. (1986) "Knowledge Representation and Utilization for Optimal Route Search," *IEEE Transactions on Systems, Man, and Cybernetics*, 454-462.

Koditschek, D.E. (1987) "Exact Robot Navigation by Means of Potential Functions: Some Topological Considerations," *Proceedings of the IEEE International Conference on Robotics and Automation*, Raleigh, NC, 1-6.

Korf, R.E. (1980) "Toward a Model of Representation Changes," *Artificial Intelligence*, 14, 41-78.

Kondo, K. and Ohtomi, K. (1991) "Motion planning in plant CAD systems," *ASME Design Automation Conference*, Miami.

Latombe, J.C. (1976) "Artificial Intelligence in Computer-Aided Design: the Tropic System," *IFIP Working Conference,* Austin, TX, edited by J.J. Allan, North Holland, 61-120.

Latombe, J.C. (1979) "Failure Processing in a System for Designing Complex Assemblies," *Sixth International Joint Conference on Artificial Intelligence (IJCAI)*, Tokyo, Japan.

Latombe, J.C. (1988) "Spatial Reasoning: From Robotics to Engineering," *Second Toyota Conference on Organization of Engineering Knowledge for Product Modeling in Computer Integrated Manufacturing*, Nagoya, edited by T. Sata, Elsevier, 83-105

Latombe, J.C. (1991) *Robot Motion Planning*, Kluwer Academic Publishers, Boston.

Laugier, C. and Germain, F. (1985) "An Adaptative Collision-Free Trajectory Planner," *International Conference on Advanced Robotics (ICAR)*, Tokyo, Japan.

182

Laumond, J.P. (1987a) "Finding Collision-Free Smooth Trajectories for a Non-Holonomic Mobile Robot," *Proceedings of the 10th International Joint Conference on Artificial Intelligence*, Milan, Italy, 1120-1123.

Laumond, J.P. (1987b). "Obstacle Growing in a Non Polygonal World," *Information Processing Letters*, 25, 41-50.

Lee, C.Y. (1961) "An algorithm for path connections and its applications," *IRE Transactions on Electronic Computers*, 346-365.

Lozano-Pérez, T. (1983) "Spatial Planning: A Configuration Space Approach," *IEEE Transactions on Computers*, C-32(2), 108-120.

Lozano-Pérez, T. (1987) "A Simple Motion-Planning Algorithm for General Robot Manipulators," *IEEE Journal of Robotics and Automation*, RA-3(3), 224-238.

Mackworth, A.K. (1977) "Consistency in networks of relation," *Artificial Intelligence*, 8, 99-118.

Mackworth, A.K. and Freuder, E.C. (1984) "The complexity of some polynomial network consistency algorithms for constraint satisfaction problems," *Artificial Intelligence* 25(1), 65-74.

Narikawa, N. et al. (1991) "An Automated Layout Design System for Industrial Plant Piping," *ASME Computer in Engineering Conference*.

Natarajan, B.K. (1989) "Some Paradigms for the Automated Design of Parts Feeders," *International Journal of Robotics Research*, 8(6), 98-109.

Nilsson, Nils J. (1980) *Principles of Artificial Intelligence*, Morgan Kaufmann Publishers, Inc., CA.

Ó'Dúnlaing, C. and Yap, C.K. (1982) "A Retraction Method for Planning the Motion of a Disc," *Journal of Algorithms*, 6, 104-111.

Ó'Dúnlaing, C., Sharir, M. and Yap, C.K. (1983) "Retraction: A New Approach to Motion Planning," 15th *FOCS*, 207-220.

Pignon, P., Hasegawa, T., and Laumond, J.P. (1991) "Optimal obstacle growing in motion planning for mobile robots," *IEEE/RSJ International Workshop on Intelligent Robots and Systems (IROS)*, Osaka, Japan.

Poirier, C.J. (1987) "Excellerator: automatic leaf cell layout agent," *Proceedings of ICCAD-87*, 176-179.

Preparata, F.P. and Shamos, M.I. (1985) *Computational Geometry: An Introduction*, Springer-Verlag, New York.

Quinlan, S. and Khatib, O. (1991) "Towards Real-Time Execution of Motion Tasks," *1991 International Symposium of Experimental Robotics*, Toulouse, France.

Reif, J.H. (1979) "Complexity of the Mover's Problem and Generalizations," *Proceedings of the 20th IEEE Symposium of Foundations of Computer Science*, 421-427.

Rimon, E. and Koditschek, D.E. (1988) "Exact Robot Navigation using Cost Functions: The Case of Distinct Spherical Boundaries in E^n," *IEEE International Conference on Robotics and Automation*, Philadelphia, PA, 1791-1796.

Rimon, E. and Koditschek, D.E. (1990) "Exact Robot Navigation in Geometrically Complicated but Topologically Simple Spaces," *IEEE International Conference on Robotics and Automation*, Cincinnati, OH, 1937-1942.

Rit, J.F. (1986) "Propagating temporal constraints for scheduling," *Proceedings for AAAI Conference*, Philadephia, PA, 383-388.

Rivest, R.L. and Fiduccia, C.M. (1982) "A 'greedy' channel router," *Proceedings 19th Design Automation Conference*, 418-424.

Sacerdoti, E. (1974) "Planning in a hierarchy of abstraction space," *Artificial Intelligence*, 5(2):115-135.

Sacerdoti, E. (1975) *A Structure for Plans and Behavior*, Technical Note 109, AI Center, SRI International.

Schwartz, J.T. and Sharir, M. (1983a) "On the Piano Movers' Problem: I. The Case if a Two-Dimensional Rigid Polygonal Body Moving Amidst Polygonal Barriers,"

Communications on Pure and Applied Mathematics, 36, 345-398.

Schwartz, J.T. and Sharir, M. (1983b) "On the 'Piano Movers' Problem: II. General Techniques for Computing Topological Properties of Real Algebraic Manifolds", *Advances in Applied Mathematics*, 4, Academic Press, 298-351.

Schwartz, J.T. and Sharir, M. (1983c) "On the 'Piano Movers' Problem: III. Coordinating the Motion of Several Independent Bodies: The Special Case of Circular Bodies Moving Amidst Polygonal Barriers," *International Journal of Robotics Research*, 2(3), 46-75.

Schwartz, J.T., Sharir, M. and Hopcroft, J. (1987) *Planning, Geometry, and Complexity of Robot Motion*, Ablex, Norwood, NJ.

Sechen, C. (1988) *VLSI Placement and Global Routing Using Simulated Annealing*, Kluwer Academic Publishers, Boston.

Sheridan, H.C. (1976) "An overview of a CASDAC subsystem-Computer-Aided Piping Design And Construction (CAPDAC)," *Naval Engineers Journal*, 87-98.

Shin, H. and Sangiovanni-Vincentelli, A. (1986) "Mighty: a 'rip-up and reroute' detailed router," *Proceedings of ICCAD-86*, 2-5.

Sifrony, S. and Sharir, M. (1987) "A New Efficient Motion Planning Algorithm for a Rod in Two-Dimensional Polygonal Space," *Algorithmica*, 1, 367-402.

Spivak, M. (1979) *A Comprehensive Introduction to Differential Geometry*, Publish or Perish, Wilmington, DE.

Stallman, R.M. and Sussman, G.J. (1977) "Forward Reasoning and Dependency-Directed Backtracking in a System for Computed-Aided Circuit Analysis," *Artificial Intelligence*, 9(2), 135-196.

Stefik, M. (1981) "Planning with constraints (MOLGEN: Part 1)," *Artificial Intelligence*, 16(2), 111-139.

Sussman, G.J. (1975) *A computer model of skill acquisition*, MIT Press, Cambridge.

Suwa, M. et al, (1990) "Development of Expert System for Piping Design in Nuclear Power Stations," *IHI Engineering Review*, 23(3), 96-103.

Tate, A. (1976) *Project Planning Using a Hierarchic Non-linear Planner*, Report No 25, Artificial Intelligence Department, University of Edinburgh.

Tenenberg, J. (1988) *Abstraction in Planning*, Ph.D. Thesis, University of Rochester, Dept. of Computer Science.

Tzeng, P.S and Sequin, C.H. "Codar: Congestion-directed general area router," *Proceedings of ICCAD-88*, 30-33.

Ullman, J.D. (1984) *Computational Aspects of VLSI*, Computer Science Press.

Unruh, A. and Rosenbloom, P.S. (1989) "Abstraction in problem solving and learning," In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, 681-687.

Vere, S.A. (1983) "Planning in time: Windows and durations for activities and goals," *IEEE Trans. Patt. Anal. Mach. Intell*, PAMI-5(3), 246-267.

Waltz, D. (1975) "Understanding line drawings of scenes with shadows," in P.H. Winston (ed.) *The Psychology of Computer Vision*, McGraw-Hill, New York, 19-91.

Waldinger, R. (1977) "Achieving several goals simultaneously," In E.W. Elcock and D.Michie (Eds.), *Machine Intelligence*, 8, Halsead/Wiley, New York.

Wangdahl, G.E. et al. (1974) "Minimum-Trajectory Pipe Routing," *Journal of Ship Research*, Vol. 18, No.1, 46-49.

Warran, D.H.D. (1974) *WARPLAN: A system for generating plans*, Memo 76, Computational Logic Department, School of Artificial Intelligence, University of Edinburgh.

Wilkins, D.E. (1984) "Domain-independent planning: Representation and plan generation," *Artificial Intelligence* 22(3), 269-301.

Wilson, R.H. and Rit, J.F. (1991) "Maintaining geometric dependencies in assembly planning," L.S. Homem de Mello and S. Lee (eds.) *Computer-Aided Assembly Planning*.

Yoshimura, T and Kuh, E.S. (1982) "Efficient Algorithms for Channel Routing," *IEEE Transactions on Computer-Aided Design*, CAD-1, 25-35.

Zhu, D. J. and Latombe, J. C. (1991a) "New Heuristic Algorithms for Efficient Hierarchical Path Planning," *IEEE Transactions of Robotics and Automation*, 7(1), 9-20.

Zhu, D. J. and Latombe, J. C. (1991b) "Mechanization of Spatial Reasoning for Automatic Pipe Layout Design," *AI EDAM*, Academic Press, 5(1), 1-20.

Zhu, Q and Freeman, H. (1988) "An Intelligent CAD System for Industrial Piping System Design," *ASME Proceedings of Computers in Engineering Conference*, San Francisco, CA, 55-59.